US006198824B1

(12) **United States Patent**
Shambroom

(10) **Patent No.: US 6,198,824 B1**
(45) **Date of Patent: Mar. 6, 2001**

(54) **SYSTEM FOR PROVIDING SECURE REMOTE COMMAND EXECUTION NETWORK**

(75) Inventor: **W. David Shambroom**, Arlington, MA (US)

(73) Assignee: **Verizon Laboratories Inc.**, Waltham, MA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/309,695**

(22) Filed: **May 11, 1999**

Related U.S. Application Data

(63) Continuation of application No. 08/799,402, filed on Feb. 12, 1997, now Pat. No. 5,923,756.

(51) **Int. Cl.**[7] **H04L 9/00**
(52) **U.S. Cl.** **380/279**; 713/168
(58) **Field of Search** 380/279; 713/168, 713/153, 156

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,313,521		5/1994	Torii et al. .
5,349,643		9/1994	Cox et al. .
5,416,842		5/1995	Aziz .
5,511,122		4/1996	Atkinson .
5,590,199		12/1996	Krajewski, Jr. et al. .
5,604,803		2/1997	Aziz .
5,764,687		6/1998	Kells et al. .
5,768,504		6/1998	Kells et al. .
5,923,756	*	7/1999	Shambroom 713/156

OTHER PUBLICATIONS

Freier, Alan O., et al., The SSL Protocol, Version 3.0, Mar. 4, 1996.
Kohl J. and Neuman, C., The Kerberos Network Authentication Service (V5), Sep. 1993.
Schneier, Bruce, Applied Cryptography, 2nd ed. (1996), pp. 566–572.
Steiner, Jenifer G., et al., "Kerberos: An Authentication Service for Open Network Systems," Mar. 30, 1988.
Kohl, John T., et al., "The Evolution of the Kerberos Authentication Service," Spring 1991, EurOpen Conference, Tromso, Norway.
MIT, Kerebos V5 Application Programming Library, Sep. 10, 1996.
MIT, Kerberos V5 Data Encryption Standard Library draft. p. 1.
MIT, Kerberos V5 Implementer's Guide, Sep. 10, 1996.
Jaspan, Barry, Kerberos Administration System KADM5 API Functional Specifications, Sep. 10, 1996.
Jaspan, Barry, KADM5 Library and Server Implementation Design, Sep. 10, 1996.
Kamens, Jonathan I., KADM5 Admin API Unit Test Description, Sep. 10, 1996.
Kamens, Jonathan I., Open V★Secure Admin Database API Unit Test Description★, Sep. 10, 1996.
MIT, Kerberos V5 Installation Guide (Release 1.0) Dec. 18, 1996.

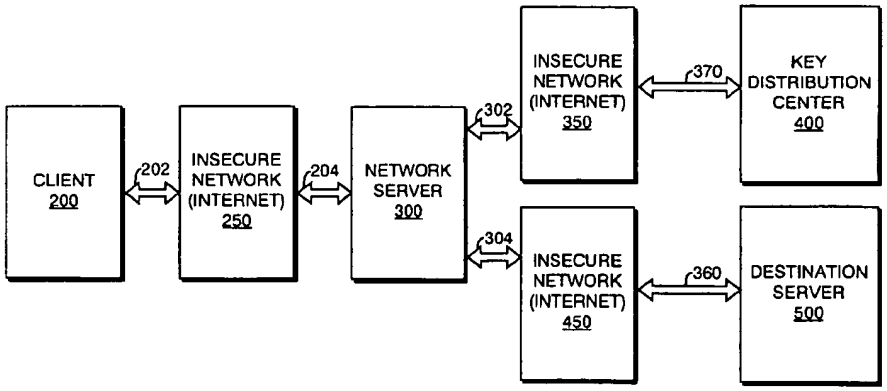
(List continued on next page.)

Primary Examiner—Salvatore Cangialosi
(74) *Attorney, Agent, or Firm*—Leonard Charles Suchyta

(57) **ABSTRACT**

A method and apparatus is disclosed for enhancing the security of a message sent through a network server from a client computer to a destination server. A secure connection for receiving and transmitting data is established between the client computer and the network server. Using client-identifying information and a secure authentication protocol, the network server may then obtain client-authentication information from a validation center. The client-authentication information is transmitted to the client and erased from the network server. The network server then receives the client-authenticating information back from the client with an accompanying message for the destination server. The network server may use the client-authenticating information to obtain permission data from the validation center for use in accessing the destination server.

13 Claims, 11 Drawing Sheets



OTHER PUBLICATIONS

MIT, Kerberos V5 System Administrator's Guide (Release 1.0), Nov. 27, 1996.

MIT, Kerberos V5 UNIX User's Guide (Release 1.0), Dec. 18, 1996.

MIT, Upgrading to Kerberos V5 from Kerberos V4 (Release 1.0), Dec. 18, 1996.

Gradient Technologies, Inc., Web Integration Strategies: Believe It Or Not—Gradient Technologies' WebCrusader, Apr. 1996, pp. 1–12.

Gradient Technologies, Inc., Developing Secure Web–based Java Applications, The Integration of Web Crusader and Net Dynamics, May 1997, pp. 1–16.

Gradient Technologies, Inc., Encryption Security in the Enterprise, Public Key/Secret Key, Jan. 1997, pp. 1–20.

InformationWeek, Spinning A Secure Web, Aug. 12, 1996 (4 pages).

Gradient Technologies, Inc., Net Crusader Product Data Sheet, NetCrusader's Distributed Services Product Line, Mar. 1997 (4 pages).

Gradient Technologies, Inc., NetCrusader Product Family Overview, Mar. 1997 (4 pages).

Gradient Technologies, Inc., NetCrusader Product Data Sheet, NetCrusader Commander, Mar. 1997 (4 pages).

Gradient Technologies, Inc., WebCrusader Product Data Sheet, WebCrusader Product Line, Mar. 1997 (4 pages).

Gradient Technologies, Inc., Web–based Applications Make the Grade at Penn State University, 1996 (2 pages).

* cited by examiner

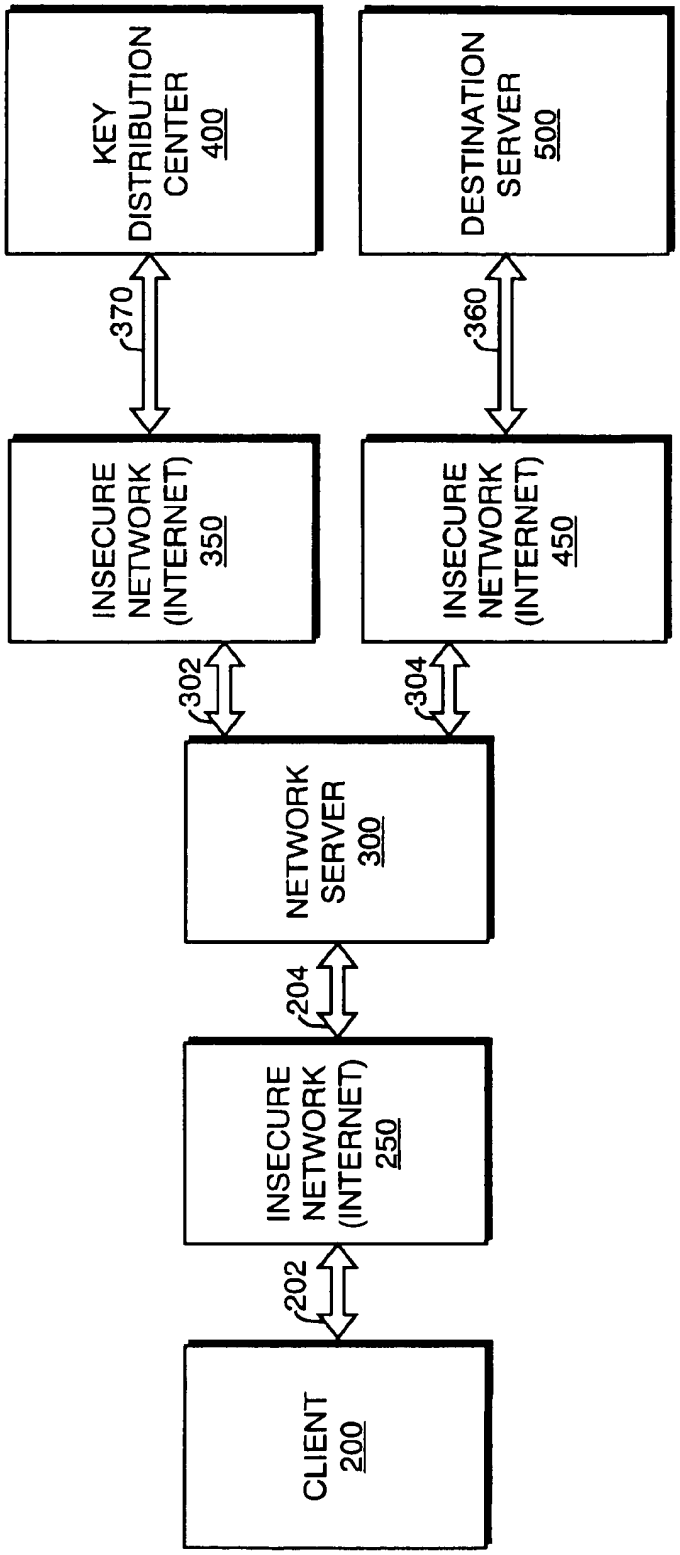

KEY DISTRIBUTION CENTER 400
DESTINATION SERVER 500
370
360
INSECURE NETWORK (INTERNET) 350
INSECURE NETWORK (INTERNET) 450
302
304
NETWORK SERVER 300
204
INSECURE NETWORK (INTERNET) 250
202
CLIENT 200

FIG. 1

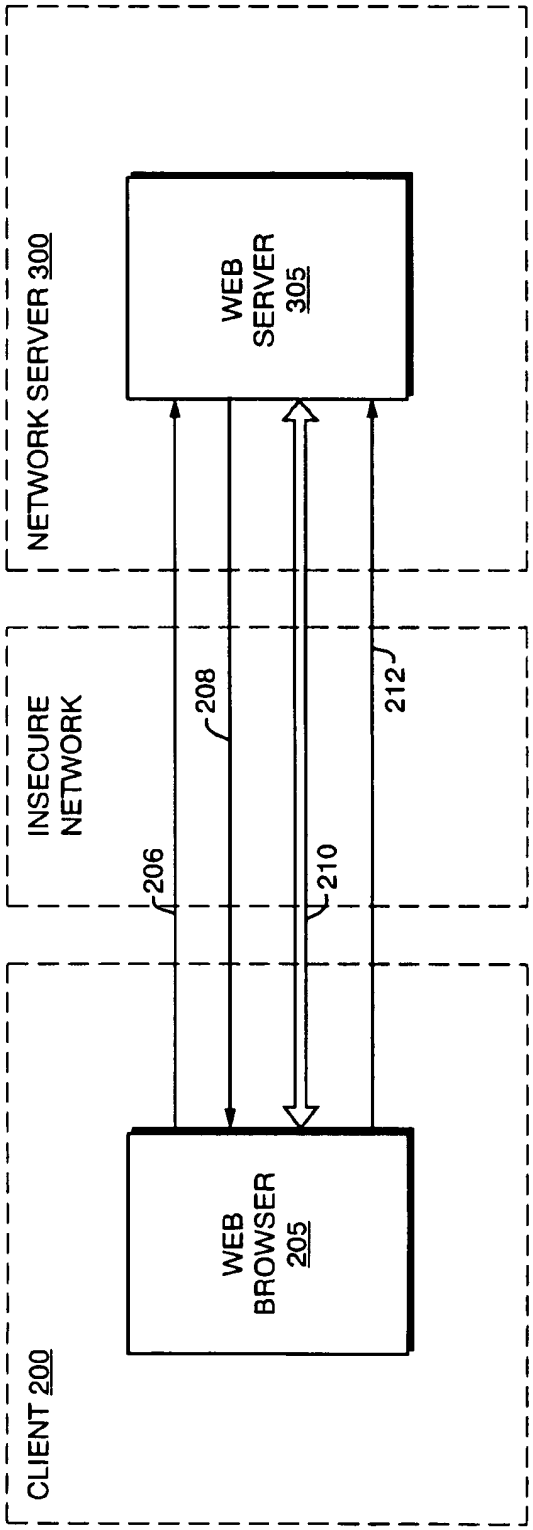

NETWORK SERVER 300
WEB SERVER 305
INSECURE NETWORK
206
208
210
212
CLIENT 200
WEB BROWSER 205

FIG. 2

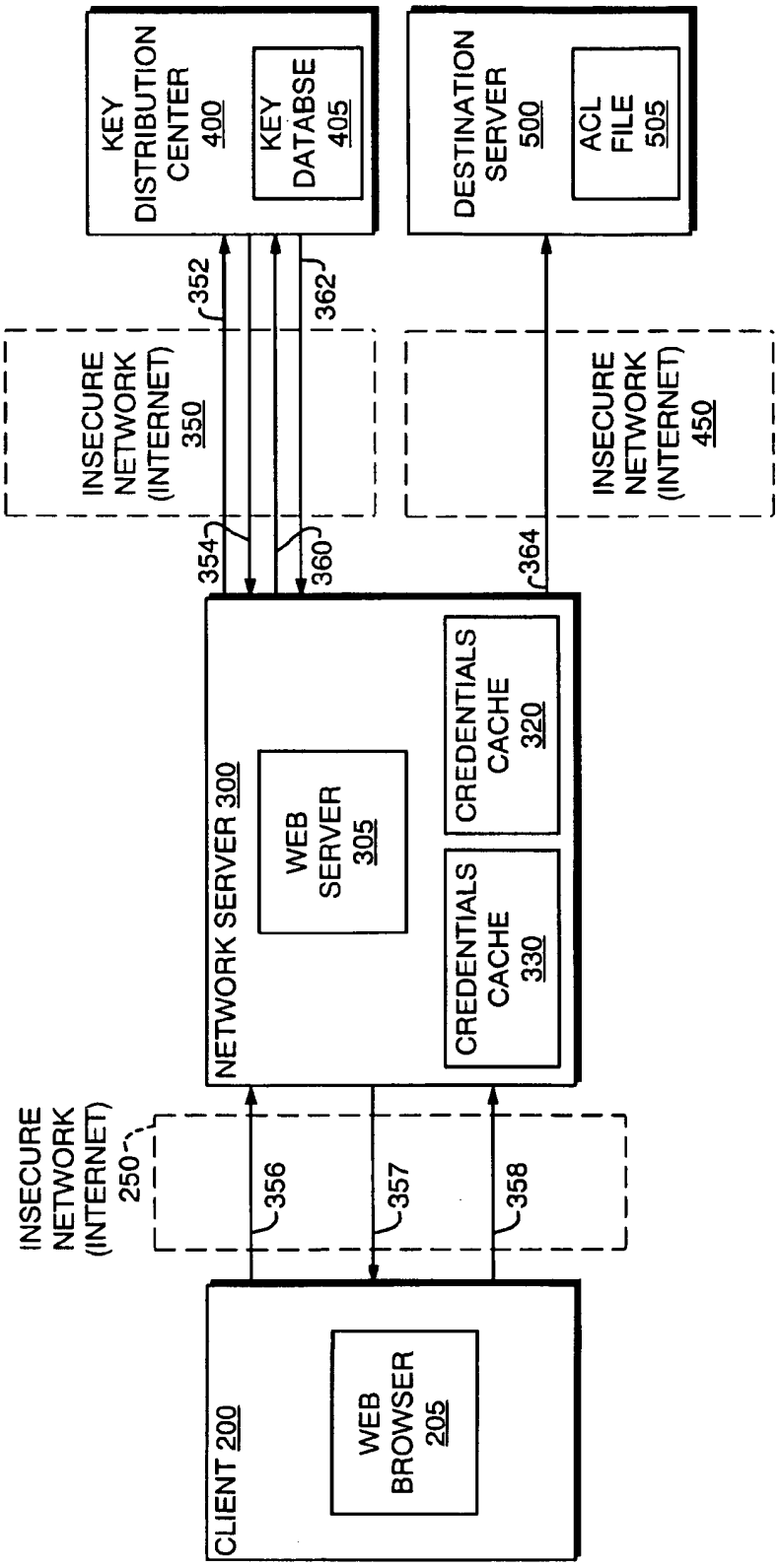

KEY DISTRIBUTION CENTER 400
KEY DATABSE 405
DESTINATION SERVER 500
ACL FILE 505
352
362
INSECURE NETWORK (INTERNET) 350
INSECURE NETWORK (INTERNET) 450
354
360
364
NETWORK SERVER 300
WEB SERVER 305
CREDENTIALS CACHE 320
CREDENTIALS CACHE 330
INSECURE NETWORK (INTERNET) 250
356
357
358
CLIENT 200
WEB BROWSER 205

FIG. 3

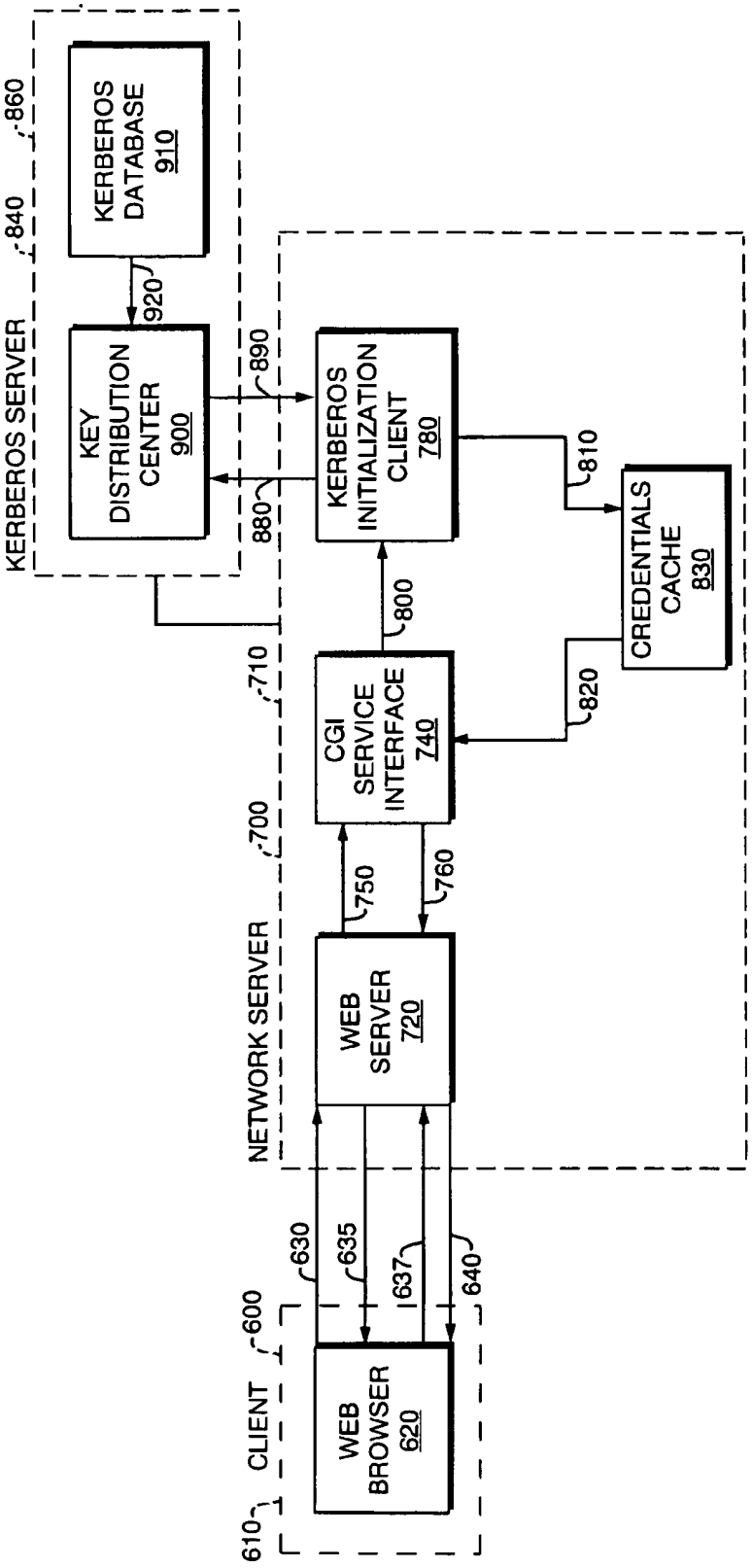

KERBEROS SERVER
840
860
KERBEROS DATABASE 910
920
KEY DISTRIBUTION CENTER 900
890
880
KERBEROS INITIALIZATION CLIENT 780
810
CREDENTIALS CACHE 830
800
820
710
CGI SERVICE INTERFACE 740
700
750
760
NETWORK SERVER
WEB SERVER 720
630
635
637
640
600
CLIENT
WEB BROWSER 620
610

FIG. 4

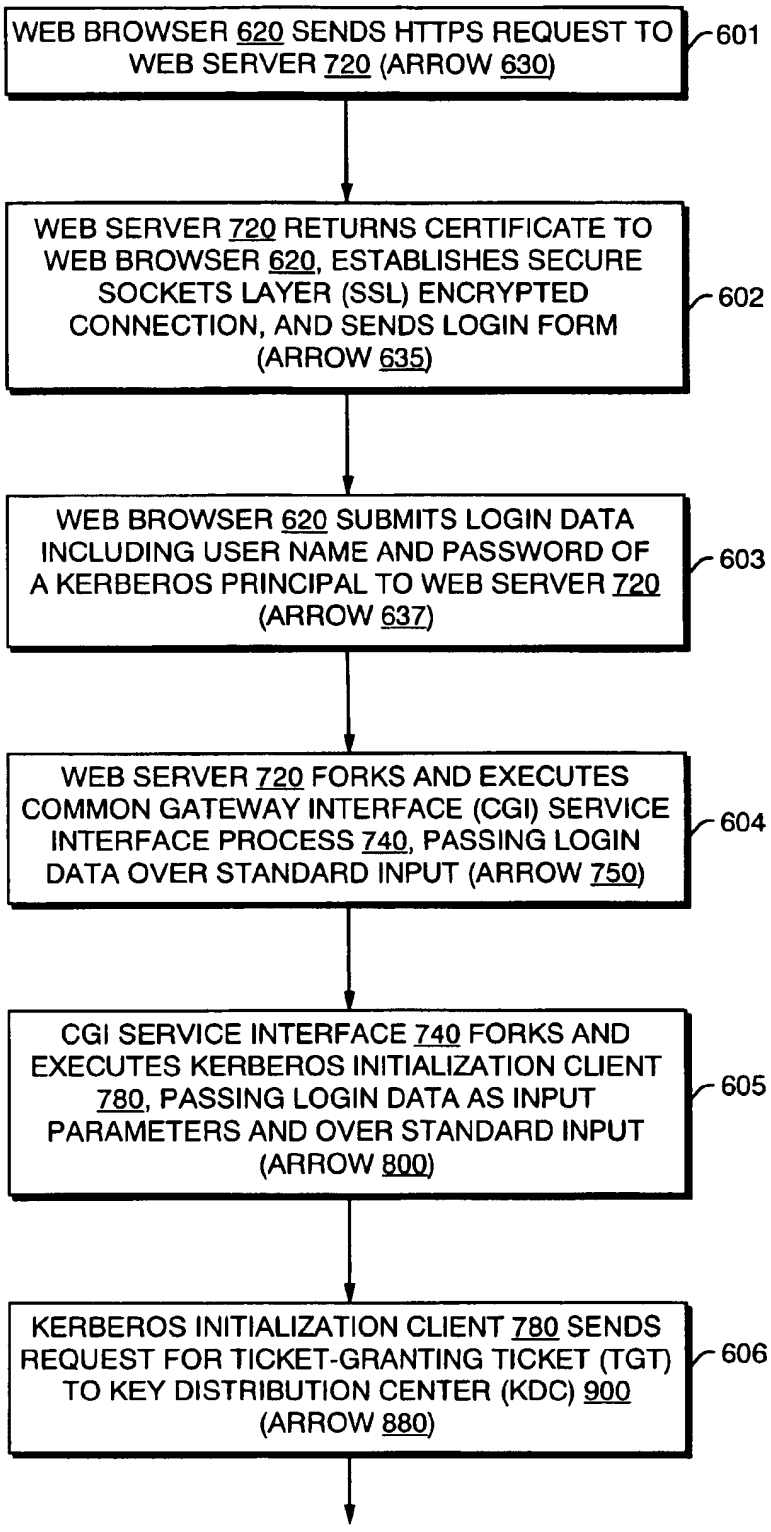

WEB BROWSER 620 SENDS HTTPS REQUEST TO WEB SERVER 720 (ARROW 630)
601
WEB SERVER 720 RETURNS CERTIFICATE TO WEB BROWSER 620, ESTABLISHES SECURE SOCKETS LAYER (SSL) ENCRYPTED CONNECTION, AND SENDS LOGIN FORM (ARROW 635)
602
WEB BROWSER 620 SUBMITS LOGIN DATA INCLUDING USER NAME AND PASSWORD OF A KERBEROS PRINCIPAL TO WEB SERVER 720 (ARROW 637)
603
WEB SERVER 720 FORKS AND EXECUTES COMMON GATEWAY INTERFACE (CGI) SERVICE INTERFACE PROCESS 740, PASSING LOGIN DATA OVER STANDARD INPUT (ARROW 750)
604
CGI SERVICE INTERFACE 740 FORKS AND EXECUTES KERBEROS INITIALIZATION CLIENT 780, PASSING LOGIN DATA AS INPUT PARAMETERS AND OVER STANDARD INPUT (ARROW 800)
605
KERBEROS INITIALIZATION CLIENT 780 SENDS REQUEST FOR TICKET-GRANTING TICKET (TGT) TO KEY DISTRIBUTION CENTER (KDC) 900 (ARROW 880)
606
TO FIG. 5A

FIG. 5

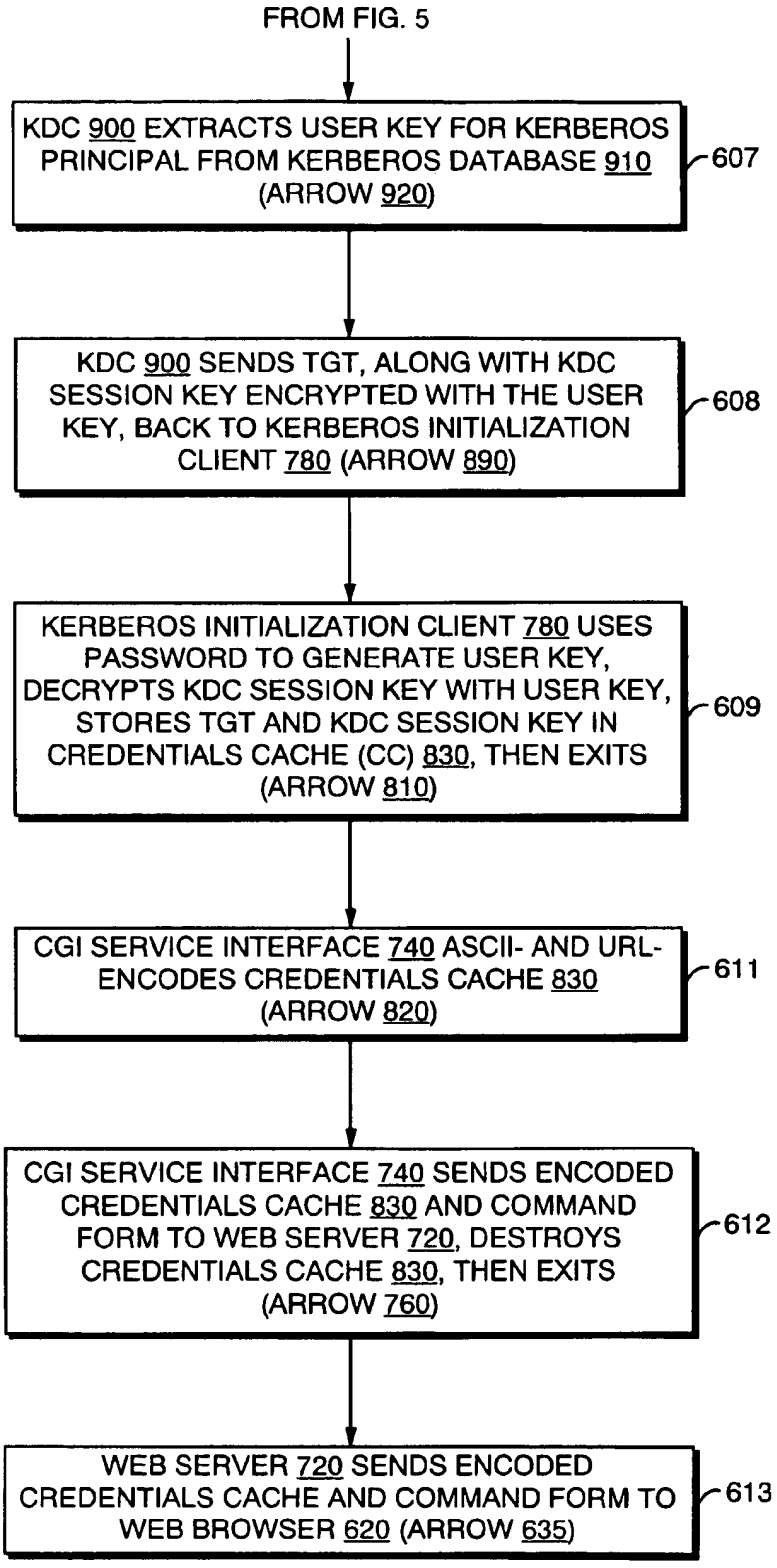

FROM FIG. 5
KDC 900 EXTRACTS USER KEY FOR KERBEROS PRINCIPAL FROM KERBEROS DATABASE 910 (ARROW 920)
607
KDC 900 SENDS TGT, ALONG WITH KDC SESSION KEY ENCRYPTED WITH THE USER KEY, BACK TO KERBEROS INITIALIZATION CLIENT 780 (ARROW 890)
608
KERBEROS INITIALIZATION CLIENT 780 USES PASSWORD TO GENERATE USER KEY, DECRYPTS KDC SESSION KEY WITH USER KEY, STORES TGT AND KDC SESSION KEY IN CREDENTIALS CACHE (CC) 830, THEN EXITS (ARROW 810)
609
CGI SERVICE INTERFACE 740 ASCII- AND URL-ENCODES CREDENTIALS CACHE 830 (ARROW 820)
611
CGI SERVICE INTERFACE 740 SENDS ENCODED CREDENTIALS CACHE 830 AND COMMAND FORM TO WEB SERVER 720, DESTROYS CREDENTIALS CACHE 830, THEN EXITS (ARROW 760)
612
WEB SERVER 720 SENDS ENCODED CREDENTIALS CACHE AND COMMAND FORM TO WEB BROWSER 620 (ARROW 635)
613

FIG. 5A

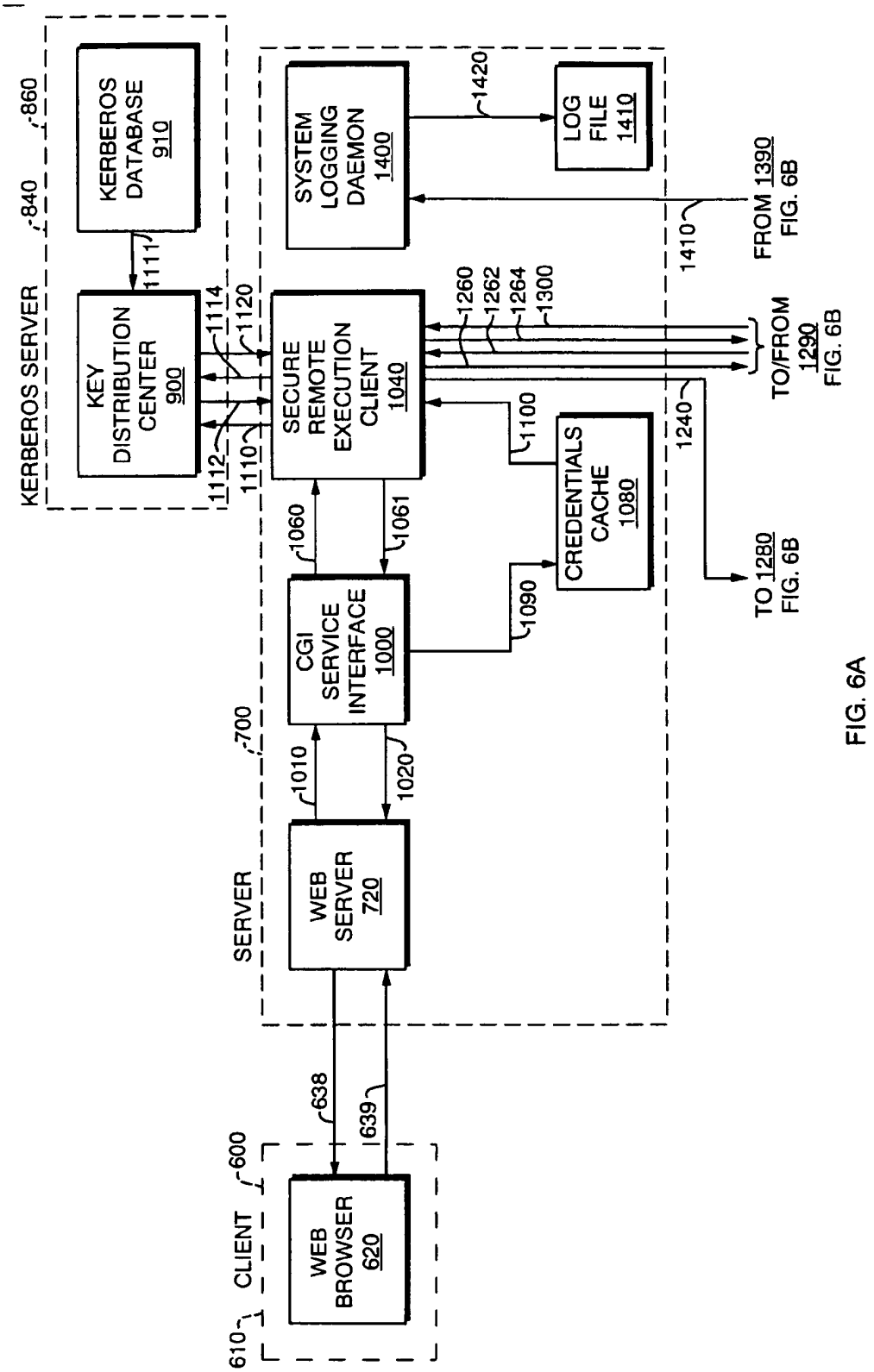

KERBEROS SERVER
840
860
KERBEROS DATABASE 910
1111
KEY DISTRIBUTION CENTER 900
1112
1114
1110
1120
SYSTEM LOGGING DAEMON 1400
1420
LOG FILE 1410
1410
FROM 1390 FIG. 6B
1260
1262
1264
1300
SECURE REMOTE EXECUTION CLIENT 1040
TO/FROM 1290 FIG. 6B
1100
CREDENTIALS CACHE 1080
1240
TO 1280 FIG. 6B
1060
1061
1090
CGI SERVICE INTERFACE 1000
700
SERVER
1010
1020
WEB SERVER 720
638
639
600
CLIENT
610
WEB BROWSER 620

FIG. 6A

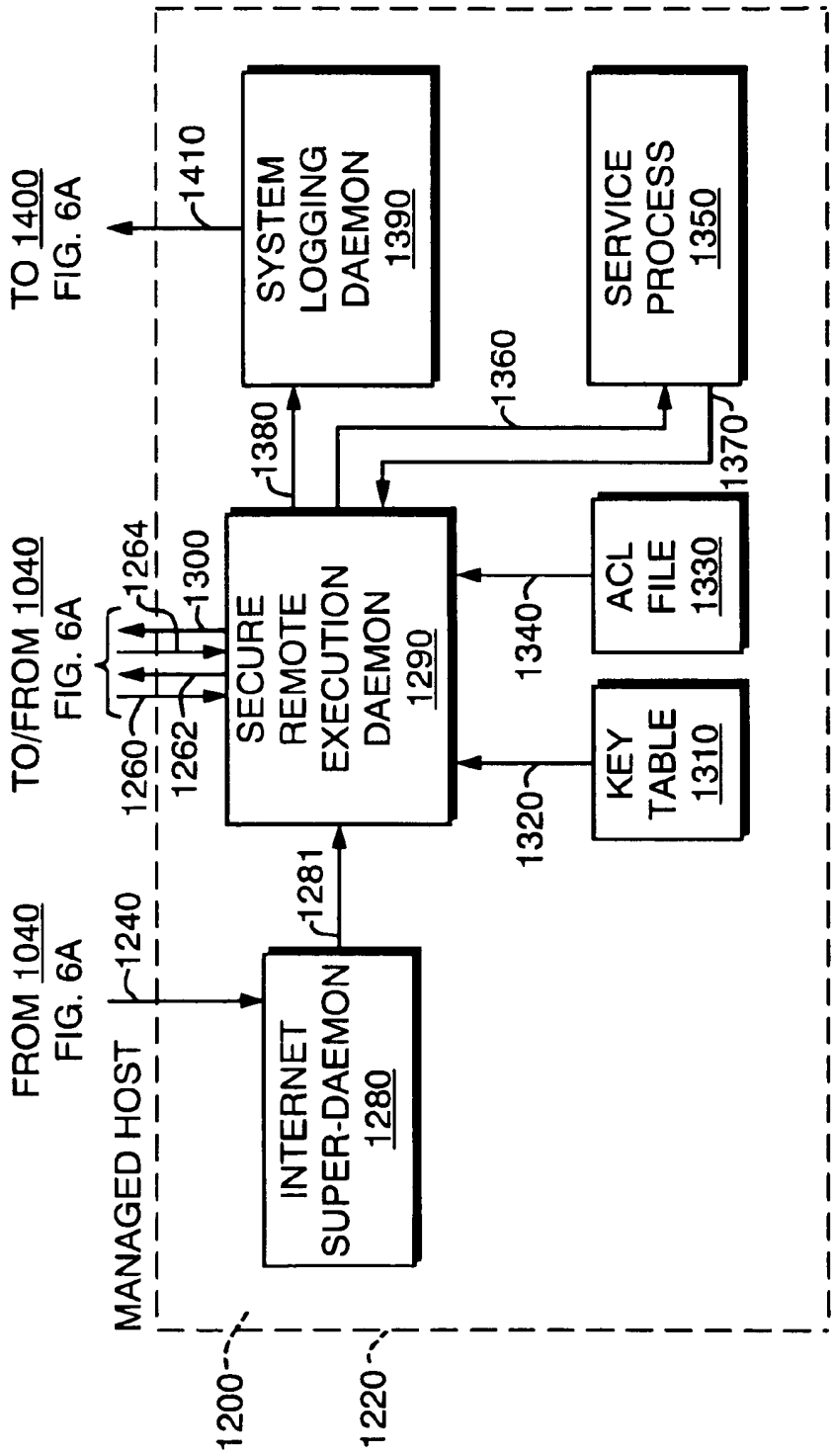

MANAGED HOST
1200
1220
INTERNET SUPER-DAEMON 1280
SECURE REMOTE EXECUTION DAEMON 1290
SYSTEM LOGGING DAEMON 1390
SERVICE PROCESS 1350
ACL FILE 1330
KEY TABLE 1310
FROM 1040 FIG. 6A
TO/FROM 1040 FIG. 6A
TO 1400 FIG. 6A
1240
1281
1260
1262
1264
1300
1320
1340
1360
1370
1380
1410

FIG. 6B

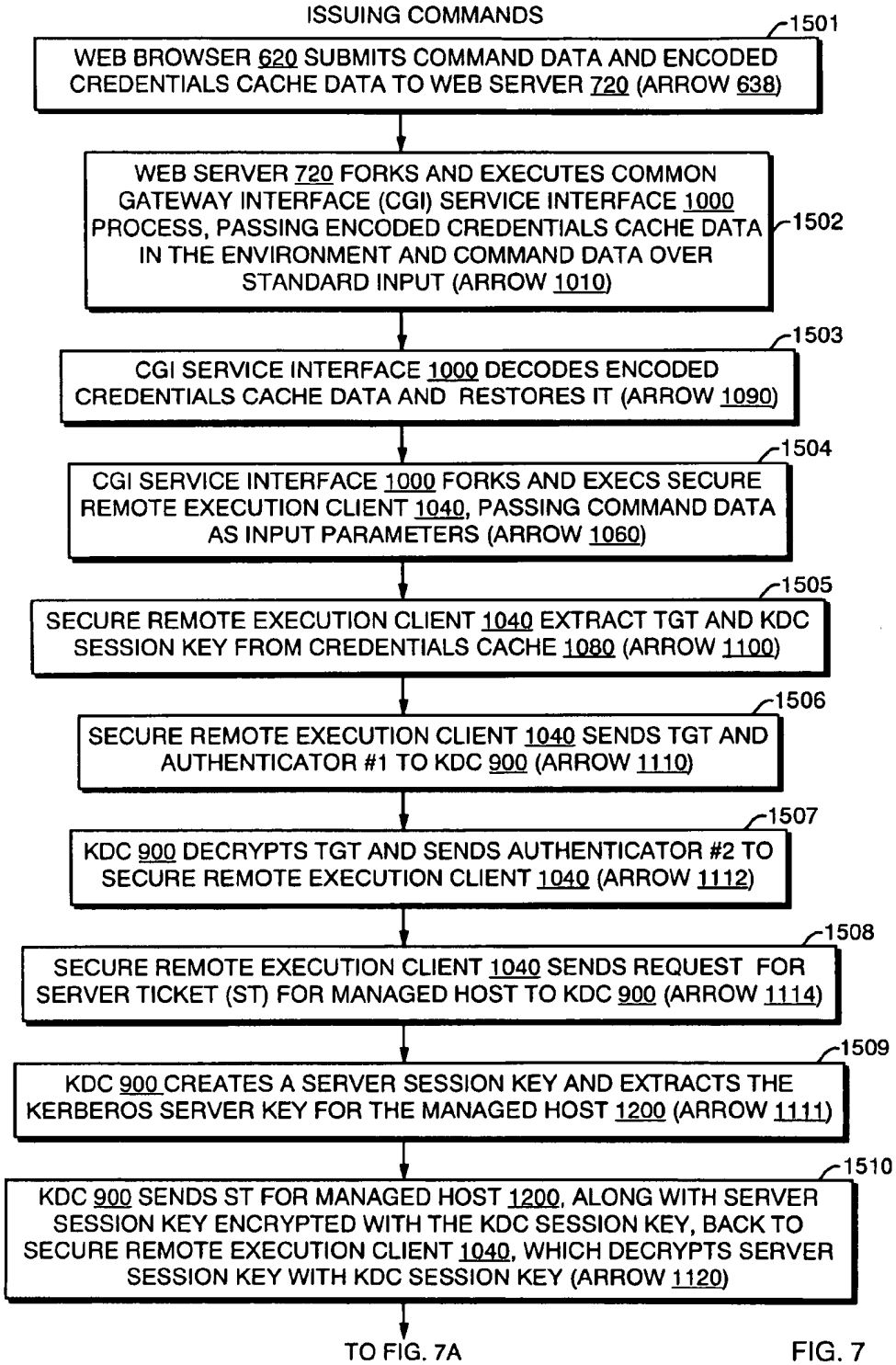

ISSUING COMMANDS
1501
WEB BROWSER 620 SUBMITS COMMAND DATA AND ENCODED CREDENTIALS CACHE DATA TO WEB SERVER 720 (ARROW 638)
WEB SERVER 720 FORKS AND EXECUTES COMMON GATEWAY INTERFACE (CGI) SERVICE INTERFACE 1000 PROCESS, PASSING ENCODED CREDENTIALS CACHE DATA IN THE ENVIRONMENT AND COMMAND DATA OVER STANDARD INPUT (ARROW 1010)
1502
1503
CGI SERVICE INTERFACE 1000 DECODES ENCODED CREDENTIALS CACHE DATA AND RESTORES IT (ARROW 1090)
1504
CGI SERVICE INTERFACE 1000 FORKS AND EXECS SECURE REMOTE EXECUTION CLIENT 1040, PASSING COMMAND DATA AS INPUT PARAMETERS (ARROW 1060)
1505
SECURE REMOTE EXECUTION CLIENT 1040 EXTRACT TGT AND KDC SESSION KEY FROM CREDENTIALS CACHE 1080 (ARROW 1100)
1506
SECURE REMOTE EXECUTION CLIENT 1040 SENDS TGT AND AUTHENTICATOR #1 TO KDC 900 (ARROW 1110)
1507
KDC 900 DECRYPTS TGT AND SENDS AUTHENTICATOR #2 TO SECURE REMOTE EXECUTION CLIENT 1040 (ARROW 1112)
1508
SECURE REMOTE EXECUTION CLIENT 1040 SENDS REQUEST FOR SERVER TICKET (ST) FOR MANAGED HOST TO KDC 900 (ARROW 1114)
1509
KDC 900 CREATES A SERVER SESSION KEY AND EXTRACTS THE KERBEROS SERVER KEY FOR THE MANAGED HOST 1200 (ARROW 1111)
1510
KDC 900 SENDS ST FOR MANAGED HOST 1200, ALONG WITH SERVER SESSION KEY ENCRYPTED WITH THE KDC SESSION KEY, BACK TO SECURE REMOTE EXECUTION CLIENT 1040, WHICH DECRYPTS SERVER SESSION KEY WITH KDC SESSION KEY (ARROW 1120)
TO FIG. 7A

FIG. 7

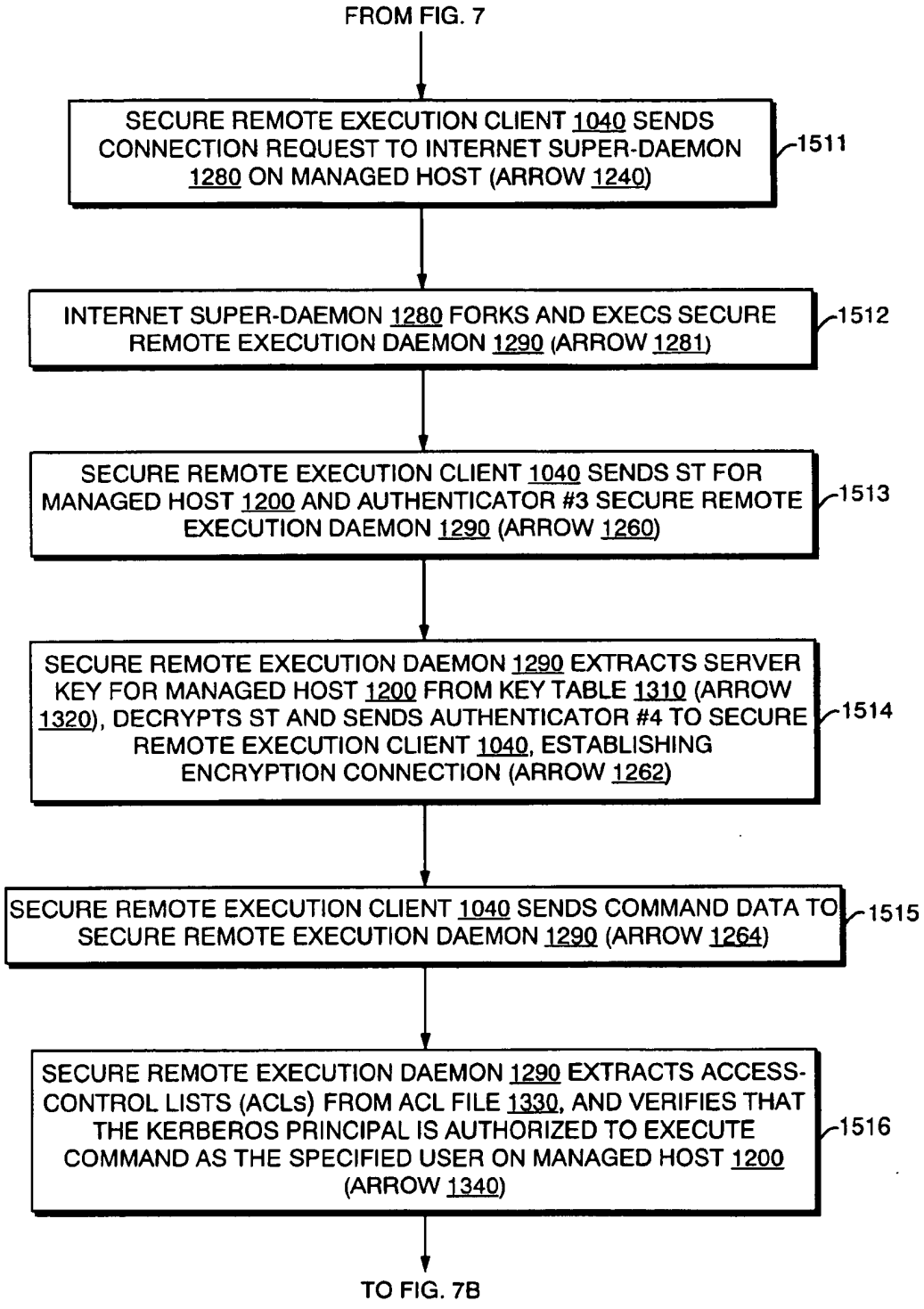

FROM FIG. 7
SECURE REMOTE EXECUTION CLIENT 1040 SENDS CONNECTION REQUEST TO INTERNET SUPER-DAEMON 1280 ON MANAGED HOST (ARROW 1240)
1511
INTERNET SUPER-DAEMON 1280 FORKS AND EXECS SECURE REMOTE EXECUTION DAEMON 1290 (ARROW 1281)
1512
SECURE REMOTE EXECUTION CLIENT 1040 SENDS ST FOR MANAGED HOST 1200 AND AUTHENTICATOR #3 SECURE REMOTE EXECUTION DAEMON 1290 (ARROW 1260)
1513
SECURE REMOTE EXECUTION DAEMON 1290 EXTRACTS SERVER KEY FOR MANAGED HOST 1200 FROM KEY TABLE 1310 (ARROW 1320), DECRYPTS ST AND SENDS AUTHENTICATOR #4 TO SECURE REMOTE EXECUTION CLIENT 1040, ESTABLISHING ENCRYPTION CONNECTION (ARROW 1262)
1514
SECURE REMOTE EXECUTION CLIENT 1040 SENDS COMMAND DATA TO SECURE REMOTE EXECUTION DAEMON 1290 (ARROW 1264)
1515
SECURE REMOTE EXECUTION DAEMON 1290 EXTRACTS ACCESS-CONTROL LISTS (ACLs) FROM ACL FILE 1330, AND VERIFIES THAT THE KERBEROS PRINCIPAL IS AUTHORIZED TO EXECUTE COMMAND AS THE SPECIFIED USER ON MANAGED HOST 1200 (ARROW 1340)
1516
TO FIG. 7B

FIG. 7A

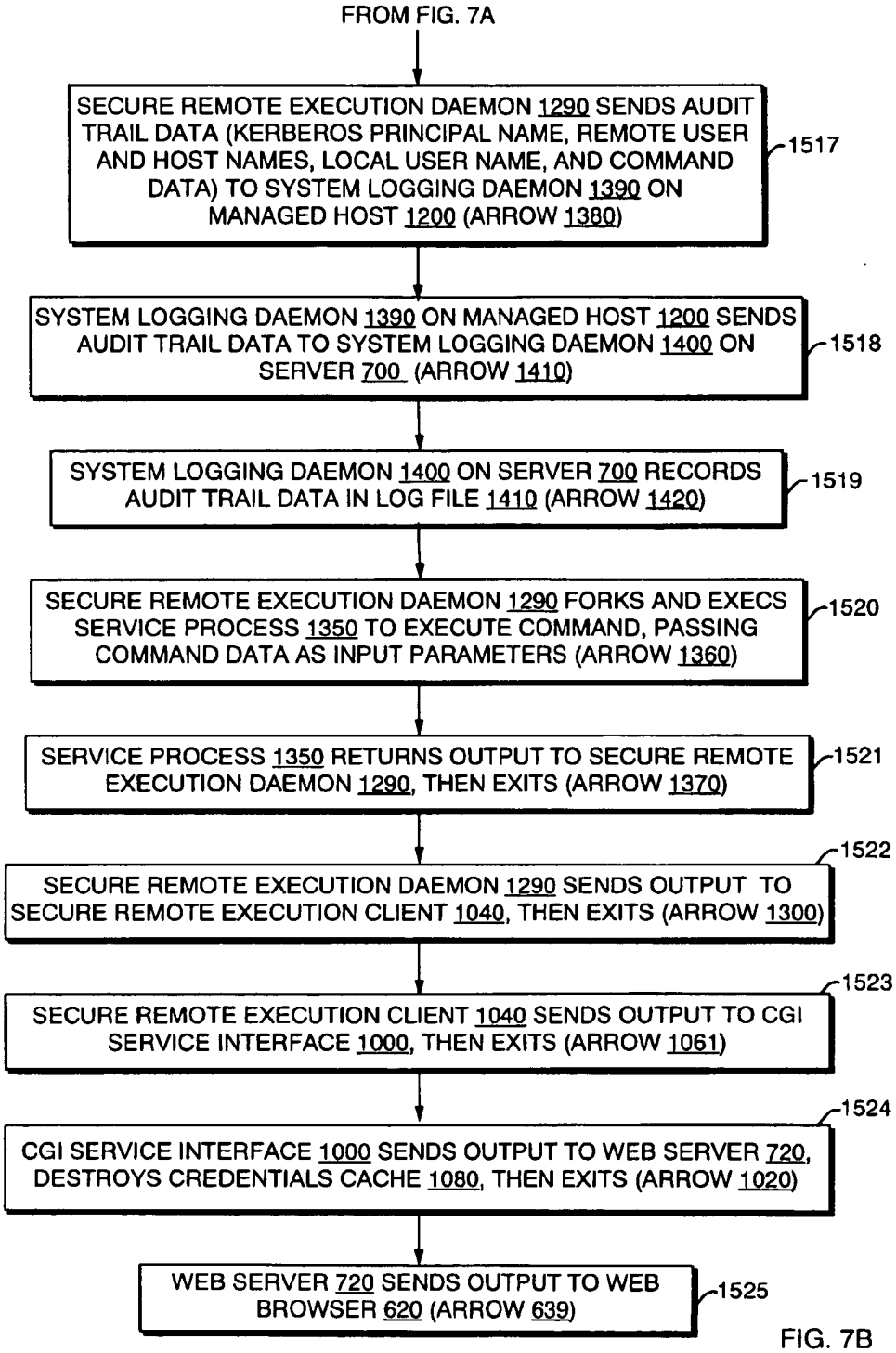

FROM FIG. 7A
SECURE REMOTE EXECUTION DAEMON 1290 SENDS AUDIT TRAIL DATA (KERBEROS PRINCIPAL NAME, REMOTE USER AND HOST NAMES, LOCAL USER NAME, AND COMMAND DATA) TO SYSTEM LOGGING DAEMON 1390 ON MANAGED HOST 1200 (ARROW 1380)
1517
SYSTEM LOGGING DAEMON 1390 ON MANAGED HOST 1200 SENDS AUDIT TRAIL DATA TO SYSTEM LOGGING DAEMON 1400 ON SERVER 700 (ARROW 1410)
1518
SYSTEM LOGGING DAEMON 1400 ON SERVER 700 RECORDS AUDIT TRAIL DATA IN LOG FILE 1410 (ARROW 1420)
1519
SECURE REMOTE EXECUTION DAEMON 1290 FORKS AND EXECS SERVICE PROCESS 1350 TO EXECUTE COMMAND, PASSING COMMAND DATA AS INPUT PARAMETERS (ARROW 1360)
1520
SERVICE PROCESS 1350 RETURNS OUTPUT TO SECURE REMOTE EXECUTION DAEMON 1290, THEN EXITS (ARROW 1370)
1521
SECURE REMOTE EXECUTION DAEMON 1290 SENDS OUTPUT TO SECURE REMOTE EXECUTION CLIENT 1040, THEN EXITS (ARROW 1300)
1522
SECURE REMOTE EXECUTION CLIENT 1040 SENDS OUTPUT TO CGI SERVICE INTERFACE 1000, THEN EXITS (ARROW 1061)
1523
CGI SERVICE INTERFACE 1000 SENDS OUTPUT TO WEB SERVER 720, DESTROYS CREDENTIALS CACHE 1080, THEN EXITS (ARROW 1020)
1524
WEB SERVER 720 SENDS OUTPUT TO WEB BROWSER 620 (ARROW 639)
1525

FIG. 7B

SYSTEM FOR PROVIDING SECURE REMOTE COMMAND EXECUTION NETWORK

This application is a continuation of U.S. Ser. No. 08/799,402 filed on Feb. 12, 1997 now U.S. Pat. No. 5,923,756 by the same inventor as this application and assigned to the same assignee.

BACKGROUND OF THE INVENTION

The present invention relates to improving the security of data transmission between computers using an insecure network, particularly to methods and systems for improving the integrity and security of messages transmitted from a client to a network server and then to a destination server or from the destination server to a network server and then to the client as part of a distributed computer system.

A distributed computer system contains multiple distinct computers, which are interconnected. One simple example of a general-purpose distributed system is a networked system containing several workstations and servers interconnected through a network. Networks are popular because they allow organizations to share information and resources. Furthermore, in a networked system, if one computer breaks, or "crashes," the others may continue to operate.

The type, cost and reliability of the manner of interconnection can be important considerations in networked systems. Large networks over relatively short distances typically use local area networks (LAN) such as an Ethernet or a Token Ring, which permit communications between a number of different computers on one or more wires. The use of modems allows computer networks to be created over a larger area, because the connections can be made over data links such as telephone lines. Wide area networks (WAN) typically use a combination of fiber optic and copper wire telephone lines as well as microwave links and satellites to connect several smaller LANs. Networks of networks are often referred to as internetworks.

Computer networks, particularly internetworks, can be vulnerable to security breaches. The degree of security of each component in the network differs, in part because each entity may be protected by varying layers of physical and operational security. Furthermore, each component or network in an internetwork may be owned or controlled by different organizations whose security practices differ widely. The interconnections between the computers may be similarly insecure. Since some part of the network may use physically insecure links, such as telephone lines or microwave links, hackers and interlopers may eavesdrop or intercept communications over the telephone line and modify them according to their wishes or copy them for later use. Interlopers who copy login and/or command information have the potential to use that information to gain access to other computers on the network.

Network security is typically based on three general concepts. For every request to do an operation, such as execute a diagnostic routine or perform a remote login, the network 1) authenticates the request; 2) controls access via access control criteria; and, 3) audits every request to detect unauthorized uses.

Authentication is the process of determining that an authorized user initiated the request and that the request was not modified improperly by an interloper on the way to the destination. One common example of authentication is the use of a password at time of login. Upon receiving a username and password from the user, a host computer

compares the password to a list of authorized usernames in an access control file, and if the password matches the password associated with that username, the host computer allows access. In the situation just described, however, it is assumed that the user and host are communicating over a secure connection; otherwise, interlopers could intercept the communications from the user to the host and steal the username and password information. The interloper could then illegally access the host at a later time by using the stolen username and password information.

In a networked system comprising multiple interconnected computers, a first computer may request service from a second or destination server through an intermediate server. This first computer is typically called a client. In order to receive service from a destination server, the client must begin by authenticating itself to the destination server. However, because the client may be communicating to the destination server over an insecure line, the client cannot simply send a password in the clear. Instead, the client and the destination server may engage in a multiple query and response exchange, constituting an authentication process, which will convince the destination server that the requesting client is an authorized user.

The prior art includes examples of encryption-based authentication processes that can be used to so authenticate a client to such a server. Such authentication processes can be based either on public-key or secret-key encryption systems. In a typical secret-key authentication scheme, each authorized party possesses a secret key, which is known only by the party and is registered with a trusted third party, or authentication server. The authentication server maintains a list of registered users and secret keys and, therefore, must be physically secure. By contrast, in a public-key authentication system, each user has a public key and a private key. The public key is posted; the private key is known only to the user. Authentication using a public-key authentication system is attractive because it does not require a secure authentication server.

One example of a secret-key based network authentication system is the trusted third-party authentication service called Kerberos. Network services and clients requiring authentication register with Kerberos and receive a secret key, where said key (or a pass phrase from which it can be derived) is known only to the user and a Kerberos host server. Kerberos also generates temporary session keys, which can be used to encrypt messages between two registered Kerberos principals (users or hosts). A typical Kerberos software package is Kerberos Version 5 from Project Athena at the Massachusetts Institute of Technology (MIT). The Kerberos authentication scheme also is discussed in J. Kohl and C. Neuman, The Network Authentication Service (V5), Request for Comments: 1510 (September 1993). Kerberos and other trusted third-party private authentication schemes can allow for speedier, secure access between two principals.

Other prior art systems have been developed to address network security issues. For example, two authentication protocols, Secure Sockets Layer (SSL) and Secure Hyper Text Transfer Protocol (S-HTTP), have been designed specifically to protect the information being transmitted across the Internet by using encryption. Both the client and the destination server must support SSL. SSL is application independent and operates at the Transport layer, meaning that it operates with application protocols such as HTTP, ftp, telnet, gopher, Network News Transport Protocol (NNTP), and Simple Mail Transport Protocol (SMTP). SSL supports several cryptographic algorithms to handle the authentication and encryption routines between the client and the server.

S-HTTP is a secure extension of HTTP, a communications protocol of the World Wide Web. S-HTTP is a publicly available protocol developed by Enterprise Integration Technologies. Unlike SSL, S-HTTP is more closely related to the HTTP protocol. Also, while SSL typically encrypts the communications link between a client and a server, S-HTTP can encrypt each message individually. In a client/server transaction under S-HTTP, the client does not need to possess a public key. Secure transactions may take place at any time, because the sender of an S-HTTP message sends its cryptographic preferences along with the message.

A current trend in distributed system development is the concept of managed hosts. In a managed host system, a client will access a network server and, via the network server, request access to a second server, which may be referred to as the remote host, or the managed host. In larger networks, the network server may be acting as a gateway and proxy for a large number of clients to access a large number of destination servers. In order for the transaction from a client to a destination server to be secure, both the transactions between the client and the network server and the transactions between the network server and the destination server should be secured by a network authentication process.

In a certificate-based authentication scheme, all entities that wish to communicate with one another must register with a third party called a certificate authority. The certificate authority verifies the identity of the registering party and issues certificates which the parties can then use to authenticate themselves to other registered parties. There are many certificate authorities offering suitable certificates of authentication including, for example, IBM's World Registry and Sun Microsystem's SunCA.

There are a number of problems associated with simply using one type of authentication process to secure the transactions between the client and network server and those between the network server and the destination server. Use of this system, for example, would require that the network server, all clients and all destination servers possess a certificate ultimately traceable to the same top-level certification authority. Furthermore, each individual user of a client system must be issued a client certificate. If the client certificates were stored on the individual workstations, the client would be restricted to using only particular workstations. If the client certificates were stored on a portable media, such as diskettes, they would be subject to loss or theft, decreasing the security of the overall network system. Moreover, client workstations may be any one of a number of different hardware devices, such as PCs or Macintosh, running a variety of different operating systems, such as UNIX or DOS, and there is no single medium supported by all the varieties of clients. In summary, use of a certificate authentication scheme between the client and the network server would be administratively difficult to support.

If Kerberos authentication for all transactions is used, each client workstation is required to possess the software necessary to communicate with the key distribution center. This approach encounters problems including that of providing many different versions of the software to support the many varieties of clients.

If one authentication scheme is used to secure transactions between the client and the network server, while another authentication scheme is used to secure transactions between the network server and the destination server, then in transactions between the client and the destination server, the network server must act as a proxy for the client, and it may sometimes be undesirable to require the network server to perform client authentication. Since, by using two different authentication schemes, the client would not be authenticating itself to the destination server directly, the network server needs to act as if it has the identity and memory of the client server. In server-to-server transactions, the user typically has logged on to the network server using a shell program. The shell program creates records on the network server that maintain a record of the user's identity and use (i.e. time and date). As long as the user is logged on, the shell logon program exists. In contrast, in a client-to-managed host transaction, the shell logon program is active on the client computer, but not on the server. The network server, instead, is interfacing with a key distribution center, or authentication server, on behalf of the client. To do this, a network server configured as a World Wide Web server creates and executes transient processes (such as when an HTTP Common Gateway Interface (CGI) request is executed) to query the key distribution center. These temporary processes must assume in some sense the identity of the user for the length of the transaction. Once their function is complete, however, these transient processes terminate and disappear, resulting in the loss of any identity or session state data they may have acquired.

When a network server does not maintain any information on a client once it has finished processing a request by the client, the server is described as stateless. A stateless file server avoids retaining client information by deriving information about files and positions within files from the request itself. A stateful server (e.g., one that stores file information in volatile memory) loses the information when the server crashes. In addition, if the client fails, the server may be unaware that the client is no longer using the space allocated to retain information needed for the transactions and may be unable to reclaim the space. In contrast, following the crash of a client or server, the stateless server need only respond to the last fully self-contained request from the client to continue the operation. In a UNIX operating environment, the UNIX processes (e.g. daemons) are sometimes stateful. Individual transient processes, however, are not persistent and, therefore, cannot maintain state information internally. There is a need, therefore, for a method of and system for increasing security of transactions involving multiple networked computers, and for increasing security of transactions involving a client that sends commands to a managed host via an intermediate server through a insecure connection such as the Internet.

There is also a need for a method of and system for increasing security of transactions involving a client, a network server, and a managed host, where the client is not restricted to one of a limited subset of devices or operating systems because of interoperability or administration concerns.

Moreover, a need exists for a method of and system for increasing security of transactions involving a client, a network server, and a managed host, where the increased security is attained by using an SSL protocol for communications between the client and the network server, a Kerberos authentication system is used to authenticate the identity of the client to the managed host and the managed host to the client, and the client communicates with the managed host through an insecure network connection such as the Internet.

Needs also exist to allow many varieties of clients to communicate with a destination server via a network server over an insecure network connection using authentication protocols and to allow transmission of data or commands

over an insecure computer network from a client to a destination server via a network server.

Another desire is for a system and method to allow necessary client information to pass to the network server with each transaction so that the network server may access the destination server on behalf of the client.

Further objects of the present invention will become apparent from the following drawings and detailed description of the preferred embodiments.

SUMMARY OF THE INVENTION

Systems and methods consistent in this invention increase security of data transmissions between a client, a network server and a managed host using an insecure network, such as the Internet. After establishing a secure network connection between a client and a network server, a secure authentication protocol is used to obtain at the network server client-authenticating information from a key distribution center. The client-authenticating information is transmitted from the network server to the client. The client-identifying information is transmitted back to the network server from the client along with a message for the destination server. Permission is obtained to access the destination server from the key distribution center over the insecure network using the secure authentication protocol. At the destination server, the authority of said client to access said destination server is validated using the message. The destination server is accessed with the message if the client's authority is properly validated.

Establishing the secure network connection between the client and the network server can use the Secure Sockets Layer (SSL) protocol. Obtaining client-authenticating information and securing the network connection between the network server and the destination server can use the Kerberos authentication protocol. Access to the destination server by authenticated users can be controlled by access control lists on the destination server.

A computer system consistent with the present invention, comprises a first computer server, such as a client, that issues commands over a network connection, and a second computer server, such as a network server, responsive to the first server and for accessing a fourth server on behalf of the client. The first and second servers can communicate via the same network operable connection therebetween. The second server also has an authentication device capable of generating an authentication request on behalf of the first server. A third computer server, such as a key distribution computer, receives the authentication request, responds to the request to authenticate the identity of the first server, and sends authentication indicator information regarding the first server back to said second server via the network. A fourth computer server, such as a managed host, is also interconnected to the network for receiving and executing the command from the first server if the network server transmits the authentication indicator information to the managed host and if the first server is authorized to access the fourth server.

BRIEF DESCRIPTION OF THE DRAWINGS

The accompanying drawings, which are incorporated in and constitute a part of the specification, illustrate presently preferred embodiments of the invention and, together with the general description given above and the detailed description of the preferred embodiments given below, serve to explain the principles of the invention.

FIG. **1** is a block diagram of one system that may be used to implement the present invention.

FIG. **2** is a more detailed block diagram of the client and network server of FIG. **1**.

FIG. **3** is a more detailed block diagram of the client, network server, key distribution center, and destination server of FIG. **1**.

FIG. **4** is a block diagram of another system that may be used to implement the present invention.

FIGS. **5**–**5***a* are flow charts showing the operation of the system of FIG. **4** in accordance with the present invention.

FIGS. **6**A–**6**B are block diagrams showing additional aspects of the system of FIG. **4**.

FIGS. *7a*–*7b* are flow charts showing the operation of the system of FIG. **6** in accordance with the present invention.

DETAILED DESCRIPTION OF THE INVENTION

A. First Embodiment

The method and apparatus useful to implement the present invention will first be discussed in general with reference to FIGS. **1**, **2**, and **3**.

As shown in FIG. **1**, the present invention uses a client workstation (indicated generally as client **200**), which can be, by way of example only, a personal computer (PC) running Microsoft Windows, Windows95, or WindowsNT, a Macintosh or a UNIX workstation. Client **200** is connected to an insecure network **250** (such as the Internet) via data link **202**. A network server **300**, which communicates with client **200** along insecure network connection **250**, can, by way of example only, be a UNIX server. Network server **300** is connected to insecure network connection **250** via data link **204** as well as a second insecure network connection **350** via suitable data link **302** and a third insecure network connection **450** via suitable data link **304**. Destination server **500** communicates with network server **300**, also through the insecure network connection **450**, via data link **360**. Destination server **500** can be, by way of example only, a UNIX server. A key distribution center (KDC) **400**, which validates requests to establish proper identity, is likewise in communication with network server **300** through data link **370** and insecure network connection **350**.

It is understood that FIG. **1** describes an exemplary network where each of the hardware components may be implemented by conventional, commercially available computer systems. Data links **202**, **204**, **302**, **360**, and **370** can be any suitable communications medium, such as, for example, data links using modems. Also, by way of example only, each computer or server can operate using an operating system such as UNIX.

Additionally, network server **300** and KDC **400** may contain information that can be used to compromise the security of the system, therefore, physical access to network server **300** and KDC **400** should be adequately controlled.

1. Establishing a secure network connection between a client and a network server

In the embodiment of FIG. **1**, client **200** and network server **300** communicate via insecure network **250**. Client **200** is connected to insecure network **250** via data link **202** which, by way of example only, may be a TCP/IP network connection. Network server **300** is connected to insecure network **250** via data link **204** which also may be a TCP/IP network connection. To enhance message privacy and integrity, client **200** and network server **300** preferably communicate using a secure authentication and/or encryption protocol to establish a secure network connection between client **200** and network server **300**. Any suitably reliable publicly available authentication protocol may be used, provided that such protocol is capable of successfully

proving the identity of network server **300** to client **200** to thereby result in confidence on the part of client **200** that future communications are with network server **300** and not some impersonating entity. The authentication protocol preferably also produces a session key that is known only to client **200** and network server **300** and which can be used to encrypt subsequent transactions between client **200** and network server **300**. One example of such an authentication protocol that has been developed specifically for use with TCP/IP Internet connections is the publicly available Secure Sockets Layer (SSL) protocol, Version 3.0, developed by Netscape Communications Corporation.

FIG. **2** shows in more detail one embodiment of the manner in which communications can be carried out between client **200** and network server **300**. As shown in FIG. **2**, client **200**, which can include a web browser **205**, initiates a request for authenticated secure access to the web server **305** of network server **300** as indicated by arrow **206**. Client **200** may be operating any publicly available web browser software package such as, for example, Netscape Navigator. Because the request may be transmitted in the clear across an insecure communications link, the request at **206** should not contain login or password information.

Web server **305** of network server **300** responds to the request at **206** by transmitting information back to web browser **205** that will be used to authenticate the identity of network server **300** to client **200** and support generation of additional information which will be used to encrypt future transmissions between client **200** and network server **300**. If, for example, an SSL transaction is employed in the system of FIG. **2**, web server **305** sends web browser **205**, as indicated by arrow **208**, a certificate that includes network server **300**'s public key and an identifier indicating a cryptographic algorithm supported by network server **300**. To properly establish the connection, network server **300** and client **200** perform a handshake process indicated at arrow **210** which, if successfully completed, provides both client **200** and network server **300** with a session key known only to network server **300** and client **200**. This session key can be used to encrypt future transactions between network server **300** and client **200**. In the handshake process of SSL, for example, client **200** creates a session key, encrypts the session key using one of the cryptographic algorithms indicated by network server **300** in the certificate and the public key sent by network server **300**, and sends the encrypted session key to network server **300**. After receiving the encrypted session key, network server **300** authenticates itself to client **200** by decrypting this session key and returning to client **200** a message encrypted with the underlying session key.

When the handshake indicated at arrow **210** is successfully completed, client **200** and server **300** continue to use the session key to encrypt future transactions. As depicted generally in FIG. **1**, the connection **202** and **204** between client **200** and server **300** are therefore protected to the degree of security achieved by the encryption algorithm.

Once an appropriately secure network connection is established between client **200** and network server **300**, server **305** now sends a login form to client **200**, and as indicated at **212**, client **200**, returns login data consisting of the name and password of a Kerberos principal to web server **305**.

2. Authenticating a client to a key distribution center and obtaining client-authenticating information from the key distribution center FIG. **3** depicts, by way of example only, the process of obtaining client-authenticating information from KDC **400** over an insecure TCP/IP network **350**, such as the Internet, that will later be used to establish that network server **300** is acting on behalf of the Kerberos user principal. Other publicly available secure authentication protocols may be used. The security of the system, however, may be enhanced further by implementing an authentication protocol that incorporates the use of timestamps. Timestamps can be used to restrict replay attacks, or the recording of some portion of an authentication protocol sequence and use of old messages at a later date to compromise the authentication protocol.

One example of a publicly available authentication protocol using timestamps is Kerberos Version 5 developed by Project Athena at MIT. The preferred embodiment as described below assumes the use of Kerberos Version 5. The details of this authentication procedure follow.

Once web server **305** receives encrypted login information from web browser **205** as indicated by arrow **356**, network server **300** passes the Kerberos user principal name of client **200** and a request for a permission indicator to KDC **400** over insecure network **350** as indicated by arrow **352**. Upon receiving the request for a permission indicator at **352**, the KDC **400** generates a KDC session key for protecting transactions between network server **300** and KDC **400**.

Using client **200**'s Kerberos user principal name received at **352**, the KDC **400** extracts client **200**'s secret key from key database **405**, which stores secret keys used by KDC **400** and other properly registered clients. Using client **200**'s secret key, the KDC **400** then encrypts one copy of the KDC session key and creates a permission indicator, which would typically include by way of example only, a timestamp, client **200**'s user name and network address, and another copy of the KDC session key. This permission indicator will be used later by client **200** to authenticate itself to KDC **400**. The permission indicator is encrypted with KDC **400**'s private key, which is known only to KDC **400**; KDC **400**, therefore, can later decrypt the permission indicator to verify its authenticity.

KDC **400** then sends both the encrypted session key and the permission indicator back to the network server **300** as indicated at arrow **354**. Network server **300** receives the encrypted information from KDC **400**, and decrypts the KDC session key using client **200**'s user key. In one embodiment, the client user key is a one-way hash of client **200**'s password and other information, so the network server is able to derive the user key by hashing client **200**'s password. Both the permission indicator and the KDC session key are stored in credentials cache **320**. Web server **305** encodes the contents of the credentials cache **320** and, as indicated at arrow **357**, sends the contents of the credentials cache **320** to web browser **205**. The authenticating information that may have resided in the network server **300** is then erased or otherwise deleted. Thereafter, in order for client **200** to continue with the transaction, client **200** will have to refresh the memory of server **300**. If a hacker or interloper managed to gain access to network server **300** while information was stored in credentials cache **320**, only the permission indicator and session key could be obtained, because the Kerberos password is destroyed after being used. This information would be of limited value, however, because the permission indicator, in the preferred embodiment, would contain a date/time stamp and would become worthless after a specified period of time, usually relatively short, has elapsed.

3. Sending a command to a destination server

Now that it has the encoded credentials cache information from cache **320**, client **200** can send this cache information along with a message, such as a command ultimately

intended for destination server **500**, to the network server **300** as indicated at arrow **358**. Network server **300** decodes the encoded credentials cache information and stores the permission indicator and KDC session key in a credentials cache **330**. Although this credentials cache **330** is not the same as credentials cache **320**, which as described above, the data therein is the same. In actuality, the information could be stored in the same location on the same physical storage device, although as a practical matter this is highly unlikely.

As indicated at arrow **360**, network server **300** now sends the permission indicator encrypted by the session key to KDC **400**, along with an authenticator and a request to access destination server **500**. This authenticator contains the Kerberos user principal name and a time stamp, encrypted using the KDC session key. KDC **400** decrypts the permission indicator using the KDC secret key to obtain the KDC session key and a validity period. If the KDC **400** decrypts successfully, the KDC is assured that the permission indicator is the same one that it issued earlier. The KDC **400** then uses the KDC session key to decrypt the authenticator to obtain the Kerberos user principal name and a time stamp. If the time stamp is within the validity period, the KDC **400** generates an access indicator. The access indicator typically would include the Kerberos user principal name, a validity period, and a server session key for use between network server **300** and destination server **500**, all of which has been encrypted with the private key of the destination server **500**. KDC **400** then sends to network server **300** the encrypted access indicator, and a copy of the server session key encrypted using the KDC session key, as indicated at arrow **362**.

Thereafter, network server **300** decrypts the copy of the server session key that is encrypted using the KDC session key. Network server **300** then encrypts the message or command, using the server session key and, as indicated at arrow **364**, sends the encrypted message along with the access indicator and a new authenticator to destination server **500** via insecure network **450**. Destination server **500** uses its own private key to decrypt and obtain the server session key.

By using the server session key, known only to destination server **500** and the network server **300**, the authenticity of the identity of client **200** can be validated at destination server **500**. The destination server **500** can then trust the integrity of the message, such as a command, from client **200**, thereby permitting access to server **500** if validation is correct. Destination server **500** can compare the identity of client **200** to a list of access control criteria (ACL) that can be stored in ACL file **505** in destination server **500**.

B. Second Embodiment

A more detailed embodiment of the present invention, in particular an embodiment using a Kerberos authentication process, is depicted in FIGS. **4** through **7**. FIG. **4**, in conjunction with the flowchart of FIGS. **5–5***a*, describes the details of a login process. Once login has been properly achieved, FIG. **6**, in conjunction with FIGS. **7–7***b*, describes the details of how a command is issued from a client to a destination server such as a managed host.

1. The Login Procedure

With reference now to FIG. **4**, client **600**, indicated generally by dotted lines **610**, includes web browser **620**. Web browser **620** communicates with network server **700**, which is indicated generally by dotted lines **710**. As will be further described below, arrows **630**, **635**, **637**, and **640** indicate the exchange of information between web browser **620** and web server **720** of network server **700**. Web server **720** exchanges information with a first CGI Service Interface **740**, as indicated by arrows **750** and **760**. CGI Service Interface **740** can be a process forked by web server **720**. As indicated by arrows **800**, **810**, and **820**, CGI Service Interface **740** in turn exchanges information with Kerberos Initialization Client **780**, which can be a process forked by CGI Service Interface **740**. Network Server **700** further includes credentials cache **830**, which receives information from Kerberos Initialization Client **780** as indicated by arrow **810** and sends information to CGI Service Interface **740** as indicated by arrow **820**.

As shown by arrows **880** and **890**, network server **700**, and in particular the Kerberos Initialization Client **780**, communicates with a Kerberos server **840**, indicated generally by dotted line **860**. In this embodiment, Kerberos server **840** includes a Key Distribution Center (KDC) **900**, which has access to Kerberos database **910** as indicated by arrow **920**. Kerberos Server **840** can be a group of processes running on the same computer as the network server **700**, or on a different computer.

The flowchart of FIGS. **5–5***a* further describe how the system of FIG. **4** accomplishes the login procedure. The term "Arrow" used in the boxes of the flowchart refers back to the corresponding numbers in FIG. **4**. Web browser **620** sends an HTTPS request to web server **720**. [Box **601**]. Web server **720** responds with a certificate to web browser **620**. This certificate contains the network server's public key and a list of one or more cryptographic algorithms that the network server supports and, by way of example only, may resemble an ITU X.**509** standard certificate. Web server **720** also establishes a Secure Sockets Layer (SSL) encrypted connection with Web browser **620**, and sends a login form to browser **620**. [Box **602**].

In response, web browser **620** submits login data back to web server **720** that would include, in this example, the user name and password of a Kerberos principal. [Box **603**].

Web server **720** executes Common Gateway Interface (CGI) Service Interface **740**. The login data is passed from web server **720** to CGI Service Interface **740** over a standard input. [Box **604**]. The CGI Service Interface **740** process is a transient process which passes login information to the Kerberos Initialization Client **780**. More specifically, the CGI Service Interface **740** executes the Kerberos Initialization Client **780**. Login data is passed as input parameters and over standard input to the Kerberos Initialization Client **780** from CGI Service Interface **740** via **800**. [Box **605**]. The Kerberos Initialization Client **780** sends a request for a ticket-granting ticket (TGT) to Key Distribution Center (KDC) **900** of Kerberos Server **840**. [Box **606**].

In other words, the Kerberos Initialization Client **780** initiates a request to the KDC **900** for a permission indicator, here, for example, the TGT. As already explained above, the permission indicator contains information that will be used during future transactions with KDC **900** for proper authentication.

KDC **900** extracts the user key for the Kerberos principal from Kerberos database **910**. [Box **607**]. In the Kerberos application, client **600**'s secret key is preferably a secure one-way hash of client **600**'s password. Then, the KDC **900** sends the TGT, along with a KDC session key encrypted with the user key, back to the Kerberos Initialization Client. [Box **608**].

The Kerberos Initialization Client **780** uses client **600**'s password to generate the user key, decrypts the KDC session key with the user key, stores the TGT and KDC session key in credentials cache **830**, and then exits. [Box **609**]. Credentials cache **830** is a data storage device used in the

processing of the transaction that makes this data available to the CGI Service Interface **740**.

CGI Service Interface **740** ASCII- and URL-encodes the credentials cache. [Box **611**]. The CGI Service Interface **740** then sends the encoded credentials cache and a command form to Web Server **720**, destroys the credentials cache, then exits. [Box **612**]. Web Server **720** sends the encoded credentials cache and the command form to Web Browser **620**. [Box **613**].

In other words, once the Initialization Client **780** stores the information in the credentials cache **830**, the Initialization Client **780** exits. Because the Initialization Client **780** embodies a transient process, all data that is contained would normally be erased. A permission indicator and KDC session key, however, are temporarily stored in the credentials cache **830**. The CGI Interface **740** extracts the contents of the credentials cache **830** and ASCII- and URL-encodes the contents. The CGI Interface **740** is also a transient process, and it is therefore necessary to extract and pass the information to web server **720** before exiting.

The web server **720** encrypts the encoded credentials cache and sends the data to the web browser **620**, as well as a command form. Once the network server **700** sends the data to the client **600**, all transient processes which handled the data exit and terminate and consequently, all authenticating information about client **600** is erased or removed. In order for client **600** to continue with the transaction, client **600** will have to refresh the memory of the server **720** and continue the second phase of the authentication process. Because there is no information relating to the transactions residing on the network server **700** during the time period in between transactions, if an unauthorized individual manages to improperly access the network server **700**, as already explained above, any information obtained would be of limited value and the integrity of the system would be retained.

2. Issuing a command

Once proper login has been accomplished as described in FIGS. **4** and **5**–**5***a*, a command can be issued from client **600** to the managed host **1200** as described in FIGS. **6** and **7**–**7***b*. Figure numbers in FIGS. **6** and **7**–**7***b* correspond to like structure and steps in FIGS. **4** and **5**–**5***a*.

With reference now to FIG. **6**, web browser **620** of client **600** communicates with web server **720** of network server **700** as indicated by arrows **638** and **639**. Web server **720** exchanges data with CGI Service Interface **1000** as indicated by arrows **1010** and **1020**. CGI interface **1000** passes command data to the Secure Remote Execution Client **1040** as indicated at arrow **1060**. The Secure Remote Execution Client **1040** is a process forked by CGI Service Interface **1000**. CGI Service Interface **1000** also passes data to credentials cache **1080** as indicated at arrow **1090**, and credentials cache **1080** in turn passes data including the TGT to the Secure Remote Execution Client **1040** as shown by arrow **1100**. Secure Remote Execution Client **1040** communicates with the KDC **900** of Kerberos Server **840** as indicated by arrows **1110** and **1120**.

The Secure Remote Execution Client **1040** can also send data to Managed Host **1200**, indicated generally by dotted lines **1220**, as shown by arrows **1240**, **1260** and **1264**. More specifically, the Secure Remote Execution Client **1040** sends data to Internet Super-Daemon **1280** as shown by arrow **1240**, and also to the Secure Remote Execution Daemon **1290** as shown by arrows **1260** and **1264**. Internet Super-Daemon **1280** is a persistent daemon process. Secure Remote Execution Daemon **1290** is a process forked by Internet Super-Daemon **1280**. Secure Remote Execution Daemon **1290** also communicates with Secure Remote Execution Client **1040** as shown by arrows **1262** and **1300**. Secure Remote Execution Daemon **1290** has access to key table **1310** as shown by arrow **1320** and also has access to ACL file **1330** as indicated by arrow **1340**. Key table **1310** is preferably a file readable only by the root user on the managed host. The Secure Remote Execution Daemon **1290** further exchanges information with the Service Process **1350**, which is a process forked by the Secure Remote Execution Daemon **1290**, as indicated by arrows **1360** and **1370**. Secure Remote Execution Daemon **1290**, as indicated by arrow **1380**, can send data to System Logging Daemon **1390**, which is a persistent daemon process. System Logging Daemon **1390** further communicates with System Logging Daemon **1400** of Server **700** as indicated by arrow **1410**. System Logging Daemon **1400**, which is a persistent daemon process, has access to log file **1410** as indicated by arrow **1420**, for purposes of making a non-volatile record of all secure remote execution activity.

With reference now to the flow charts of FIGS. **7**–**7***b*, the system of FIG. **6** operates in the following manner. The term "Arrow" used in the boxes of the flowchart refers back to the corresponding numbers in FIG. **6**. Web browser **620** submits command data and encoded credentials cache to web server **720**. [Box **1501**]. Web server **720** executes CGI Service Interface **1000**, and passes the encoded credentials cache in the environment and command data over standard input from web server **720** to CGI Interface **1000**. [Box **1502**].

CGI Service Interface **1000** decodes the encoded credentials cache and restores it to a credentials cache **1080**. [Box **1503**]. CGI Service Interface **1000** executes the Secure Remote Execution Client **1040**, passing command data as input parameters from CGI Service Interface **1000** to Secure Remote Execution Client **1040**. [Box **1504**]. The Secure Remote Execution Client **1040** extracts the TGT and KDC session key from credentials cache **1080**. [Box **1505**].

Then, the Secure Remote Execution Client **1040** sends the TGT and an authenticator #1 to KDC **900**. [Box **1506**]. The KDC **900** decrypts the TGT and sends authenticator #2 to Secure Remote Execution Client **1040**. [Box **1507**]. Secure Remote Execution Client **1040** then sends a request for a server ticket (ST) for Managed Host **1200** to KDC **900**. [Box **1508**]. KDC **900** creates a server session key and extracts the Kerberos server principal key for Managed Host **1200** from Kerberos database **910**. [Box **1509**]. KDC **900** creates a Kerberos ST, for Managed Host **1200** and then sends the ST, along with the server session key encrypted with the KDC session key, back to Secure Remote Execution Client **1040**, which decrypts the server session key with the KDC session key. [Box **1510**]. Then, the Secure Remote Execution Client **1040** sends the connection request to Internet Super-Daemon **1280** of Managed Host **1200**. [Box **1511**].

Internet Super-Daemon **1280** forks and executes the Secure Remote Execution Daemon **1290**, passing command line parameters specifying encryption requirements. [Box **1512**]. The Secure Remote Execution Client **1040** sends the ST for Managed Host **1200** and authenticator #3 to Secure Remote Execution Daemon **1290**. [Box **1513**]. The Secure Remote Execution Daemon **1290** extracts the server key for Managed Host **1200** from key table **1310**, decrypts the server ticket and sends authenticator #4 to Secure Remote Execution Client **1040**, establishing an encrypted connection. [Box **1514**]. Secure Remote Execution Client **1040** then sends command data to Secure Remote Execution Daemon **1290**. [Box **1515**]. The Secure Remote Execution Daemon **1290** also extracts access-control lists (ACLs) from ACL file **1330**, and verifies that the Kerberos principal is

authorized to execute the command as the specified user on Managed Host **1200**. [Box **1516**].

The Secure Remote Execution Daemon **1290** also sends audit trail data (such as, for example, the Kerberos principal name, remote user and host names, local user name, and command data) to System Logging Daemon **1390** on Managed Host **1200**. [Box **1517**]. This is to provide a record of all secure remote execution activity. In turn, the System Logging Daemon **1390** can send audit trail data to System Logging Daemon **1400** on Server **700**. [Box **1518**]. The System Logging Daemon **1400** records audit trail data in log file **1410**. [Box **1519**].

The Secure Remote Execution Daemon **1290** executes Service Process **1350** to execute the command and passes command data as input parameters. [Box **1520**]. The Service Process **1350**, which is a process forked by Secure Remote Execution Daemon **1290**, returns output to Secure Remote Execution Daemon **1290**, and then exits. [Box **1521**]. The Secure Remote Execution Daemon **1290** sends output to Secure Remote Execution Client **1040**, and then exits. [Box **1522**]. The Secure Remote Execution Client **1040** sends output to CGI Service Interface **1000**, and then exits. [Box **1523**]. The CGI Service Interface **1000** sends output to Web Server **720**, destroys credentials cache **1080** and, then exits. [Box **1524**]. Web Server **720** then sends output to Web Browser **620**. [Box **1525**]. This allows the user at the client system to see the results of the command that was executed.

It should be understood that more than one server and client can be used, and that this invention is equally applicable to multiple clients and multiple destination servers.

As used herein, it is understood that the term "secure," as applied to network server **300**, destination server **500**, and KDC **400**, means that information stored in the servers is accessible under normal, expected operating conditions only by suitably authorized individuals.

While there have been shown what are presently considered to be preferred embodiments of the invention, it will be apparent to those skilled in the art that various changes and modifications can be made herein without departing from the scope of the invention as defined by the appended claims.

What is claimed is:

1. A system having a key distribution center and having improved security for a message sent over an insecure network from a client computer to a destination server via a network server, the system comprising:

a security protocol interface configured to establish communication over the insecure network with a first type of security between the client computer and the network server;

an authentication protocol interface configured to obtain first client-authenticating information from the key distribution center and to provide the first client-authenticating information obtained to the network server to establish communication over the insecure network with a second type of security;

the network server configured to transmit the first client-authenticating information to the client computer;

the client computer configured to transmit the message and the first client-authenticating information to the network server; and

the network server configured to obtain permission from the key distribution center to access the destination server over the insecure network using second client-authenticating information.

2. The system of claim **1**, further comprising:

a validation program at the destination server configured to use the message and the second client-authenticating information to validate authorization of the client computer to access the destination server.

3. The system of claim **1**, wherein the network server is configured to erase the first client-authenticating information from the network server after transmitting the first client-authenticating information to the client computer.

4. The system of claim **1**, wherein the security protocol interface utilizes a secure sockets layer protocol.

5. The system of claim **1**, wherein the second type of security is provided using a Kerberos protocol.

6. The system of claim **1**, wherein the first client-authenticating information is stored in a credentials cache.

7. The system of claim **5**, wherein the credentials cache is stored in non-volatile storage.

8. A network comprising a client computer, a key distribution center, and a destination computer capable of being operatively coupled to a network server of the network, the network comprising:

a client network interface of the network server configured to receive client-identifying information from the client computer;

a security protocol network interface of the network server configured to provide at least a portion of the client-identifying information to the key distribution center and to receive first client-authenticating information from the key distribution center;

a pass-through interface of the network server configured to transmit at least a portion of the client-identifying information and a portion of first client-authenticating information to the client computer via the client networking interface, the pass-through interface configured to receive the at least a portion of the client-identifying information and the portion of the first-client-authenticating information in order to obtain second client-authenticating information from the key distribution center; and

a destination computer network interface of the network server configured to communicate with the destination computer using at least a portion of the second client-authenticating information.

9. The network of claim **8**, wherein the network server stores at least a portion of the client-identifying information and of the first client-authenticating information in a credentials cache.

10. The network of claim **8**, wherein the pass-through interface discards the first client-authenticating information.

11. The network of claim **8**, wherein the client network interface further comprises:

a web server configured to receive the client-identifying information from the client computer;

a network server key data base, the network server key data base having a key, the key associated with a public-private key pair of a cryptographic algorithm;

a decryptor, the decryptor configured to decipher a session key generated and enciphered by the client computer using a public key of the public-private key pair of the cryptographic algorithm; and

an encryptor configured to encipher an authenticating message using the session key and the cryptographic algorithm.

12. The network of claim **8**, wherein the client network interface uses at least in part a secure sockets layer protocol.

13. The network of claim **8**, wherein the security protocol network interface uses at least a part of a Kerberos authentication protocol.

* * * * *

US006041357A

United States Patent [19]

Kunzelman et al.

[11] Patent Number: 6,041,357

[45] Date of Patent: Mar. 21, 2000

[54] **COMMON SESSION TOKEN SYSTEM AND PROTOCOL**

[75] Inventors: **Kevin Kunzelman**, San Francisco; **Sterling Hutto**, Sausalito, both of Calif.

[73] Assignee: **Electric Classified, Inc.**, San Francisco, Calif.

[21] Appl. No.: **08/796,260**

[22] Filed: **Feb. 6, 1997**

[51] **Int. Cl.**[7] **H04L 9/00**; G06F 13/14

[52] **U.S. Cl.** **709/228**; 380/49; 380/25; 395/187.01

[58] **Field of Search** 395/186, 187.01, 395/188.01, 200.48, 200.59, 200.33, 200.57; 380/49, 25

[56] **References Cited**

U.S. PATENT DOCUMENTS

5,224,163	6/1993	Gasser et al.	380/30
5,542,046	7/1996	Carlson et al.	395/186
5,689,638	11/1997	Sadovsky	395/188.01
5,706,349	1/1998	Aditham et al.	380/257.01
5,708,780	1/1998	Levergood et al.	395/200.59
5,740,242	4/1998	Minor et al.	380/49
5,774,670	6/1998	Montulli	395/200.57

OTHER PUBLICATIONS

Yoshida, H.; "The Age of Customized Web Site", *Web Developer* (Winter 1996) Tech Spotlight, located at http://W3.COM/CGI-BIN/W3 CPM/PWS/W3 . . . JhbIdxBW_F92-XXfDgM-wZ8WxcC1mT666J, pp. 1-5.

"IETF Hypertext Transfer Protocol (HTTP) Working Group" located at http://www.ecs.uci.edu/pub/ietf/http/, pp. 1-5.

A. Hutchinson, et al., "An Extension of the HTTP Authentication Scheme to Support Server Groups", located at http://www.ics.uci.edu/pub/ietf/ht . . . ft-trommler-http-ext-groups-00.txt, pp. 1-10.

P.M. Hallam-Baker, et al., "Session Identification URI", located at http://www.es.net/pub/internet-drafts/draft-ietf-http-session-id-00.txt, pp. 1-10.

D.M. Kristol, et al., "Proposed HTTP State Management System", located at http://www.research.att.com/~dmk/cookie-2.9.txt, pp. 1-17.

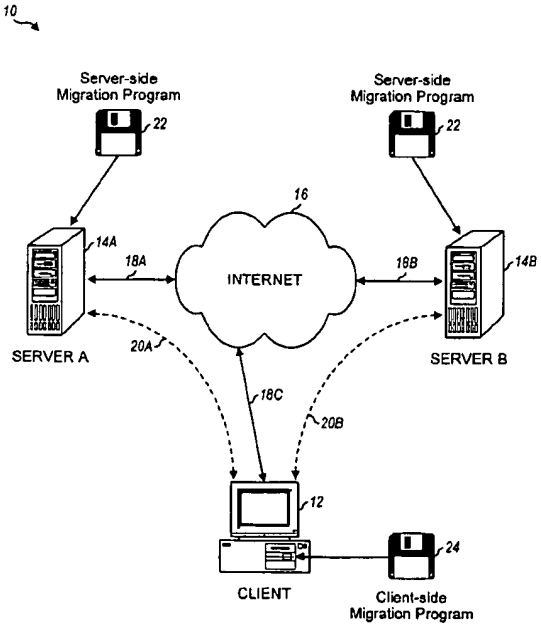
Primary Examiner—Pinchus M. Laufer

Attorney, Agent, or Firm—Philip H. Albert; Townsend and Townsend Crew

[57] **ABSTRACT**

An improved session control method and apparatus includes a client which establishes a session with a first server such that the first server can identify the client. When the client wishes to migrate from the first server to a second server, the client requests a session token from the first server. The session token is a data element generated by the first server which is unique over the client-server network being navigated and identifies the particular session with the first server. The session token is preferably a difficult to forge data element, such as a data element digitally signed using the private key of the first server. The session token is passed from the client to the second server to initiate migration to the second server. If session data is too bulky to be passed as part of the session token, the second server may use data from the session token to formulate a request to the first server for additional data needed to handle the state of the session. To minimize the transmission of data, the second server might maintain a version of the bulk session data and only request an update to the version of the data indicated in the session token.

13 Claims, 3 Drawing Sheets



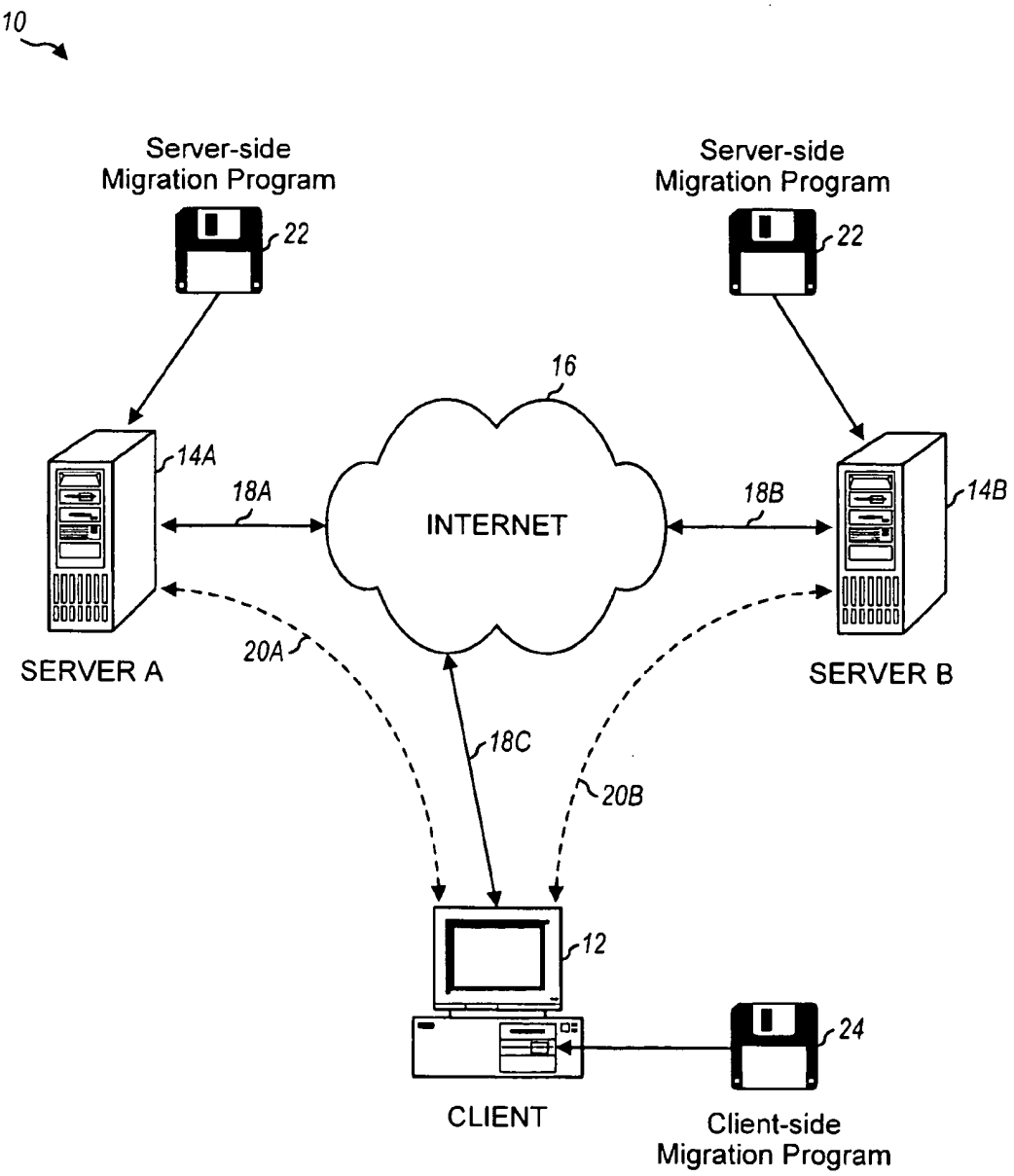

10
Server-side
Migration Program
22
Server-side
Migration Program
22
16
14A
18A
INTERNET
18B
14B
SERVER A
20A
SERVER B
18C
20B
12
24
CLIENT
Client-side
Migration Program

FIG. 1

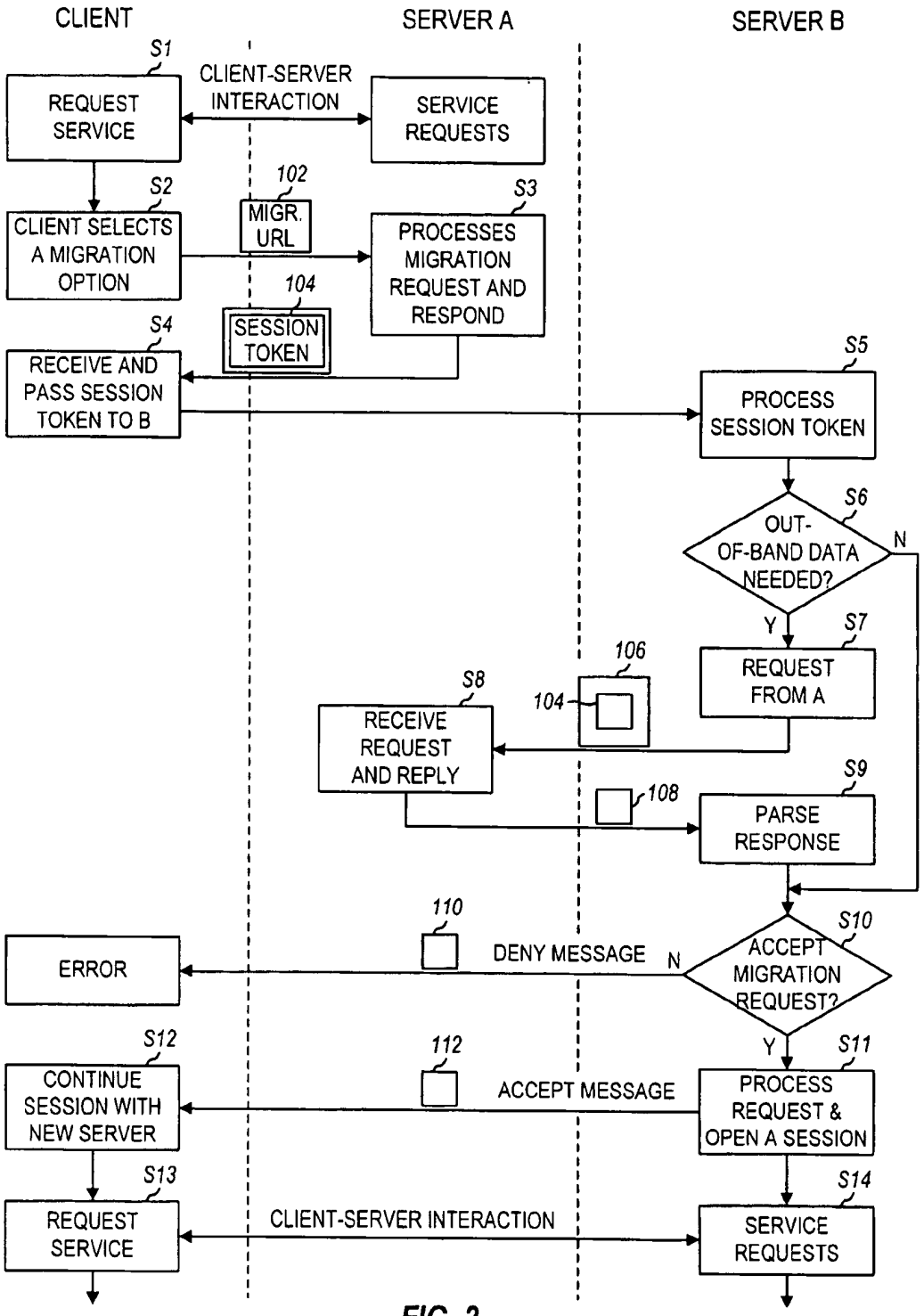

CLIENT
SERVER A
SERVER B
S1
REQUEST SERVICE
CLIENT-SERVER INTERACTION
SERVICE REQUESTS
102
MIGR. URL
S2
CLIENT SELECTS A MIGRATION OPTION
S3
PROCESSES MIGRATION REQUEST AND RESPOND
104
SESSION TOKEN
S4
RECEIVE AND PASS SESSION TOKEN TO B
S5
PROCESS SESSION TOKEN
S6
OUT-OF-BAND DATA NEEDED?
N
Y
S7
REQUEST FROM A
106
104
S8
RECEIVE REQUEST AND REPLY
108
S9
PARSE RESPONSE
110
ERROR
DENY MESSAGE
N
S10
ACCEPT MIGRATION REQUEST?
Y
S12
CONTINUE SESSION WITH NEW SERVER
112
ACCEPT MESSAGE
S11
PROCESS REQUEST & OPEN A SESSION
S13
REQUEST SERVICE
CLIENT-SERVER INTERACTION
S14
SERVICE REQUESTS

FIG. 2

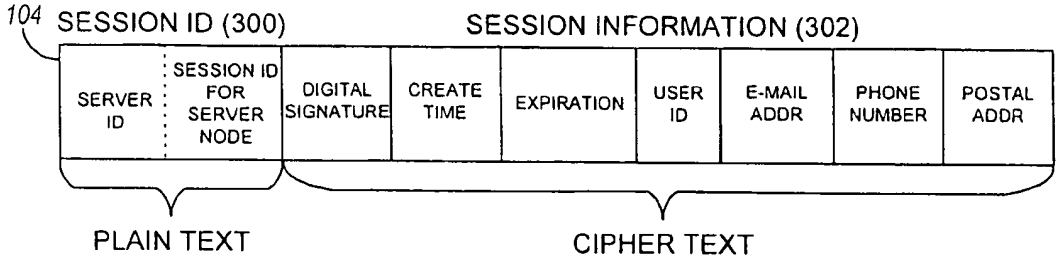

104
SESSION ID (300)
SESSION INFORMATION (302)
SERVER ID
SESSION ID FOR SERVER NODE
DIGITAL SIGNATURE
CREATE TIME
EXPIRATION
USER ID
E-MAIL ADDR
PHONE NUMBER
POSTAL ADDR
PLAIN TEXT
CIPHER TEXT

FIG. 3

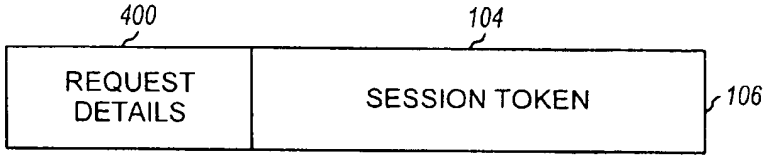

400
104
REQUEST DETAILS
SESSION TOKEN
106

FIG. 4

COMMON SESSION TOKEN SYSTEM AND PROTOCOL

BACKGROUND OF THE INVENTION

The present invention relates to the field of client-server protocols. More specifically, one embodiment of the invention provides for seamless migration of a session from one independent cooperating server node to another.

The global Internet has emerged as a model of a distributed network of networks. One application for which the Internet is increasingly used is to interconnect hypertext transfer protocol (HTTP) servers to clients. The hypertext transfer protocol is used to serve documents containing hypertext references to client "browsers" on client systems connected to the Internet. The documents served by an HTTP server will often have hypertext references embedded in the document which refer to documents on the HTTP server or documents on a completely different server. With such an arrangement, millions of documents have been linked to form the World Wide Web, which is an allusion to the fact that these hypertext links might look like a spider web if a diagram of the documents and the links were drawn.

Hypertext, as such, was used in other contexts, such as help documentation, but those uses generally connected to a central document repository or even a single file with references to locations internal to the file. What made the World Wide Web more interesting and complex is the fact that a link in a first document stored on a first server might refer to a document on a second server where the author of the first document and the system operator of the first server had no editorial or system control over the second document or the second server.

This independence of servers did not inhibit the navigation of a client from server to server, since the reference for the link to be followed was fully contained in the document where the reference was made. These self-contained references to information are referred to as "uniform resource locators" (URLs) because the reference was all that was needed to locate the "linked-to" document. Since the URL is a static data element, it is the same for every client, so the linked-to server could not identify who the client was. Several solutions to this shortcoming have been in common use.

The most common solution is to eliminate the need to know who the client is. If the identity of the client is not needed, an HTTP server will serve documents to any client which requests a page. Of course, this solution is only practical where the author of the linked-to document does not want to place any restrictions on who may view the page. If the author of the linked-to page wants to place restrictions, the client will have to enter into a "session" with the server where the client is first authenticated prior to the server allowing access to the server.

One consequence of such an architecture is that, without some session control, the clients and servers must operate in a "stateless" manner, i.e., not tracking client identity, or any other variables, from request to request. Session control, when used, is usually done by a server requesting a login name and a password from the client prior to serving documents. Where all the documents are stored on a single server or a centrally controlled cluster of servers, session control only need occur when the client first visits the server. If a link from one document references a document on the same server, the session can be seamlessly continued from the perspective of the client. However, if the link is to a document on a second server not commonly controlled with the first server, the second server will interrupt the navigation process to require authorization information from the client. The usefulness of hypertext documents is greatly diminished if a user must enter a new login name and password each time a link is taken.

Session control for a single document is known. A URL for a document available in a sessionless environment might be a character string of the form:

http://server.host.domain/directory/subdir/file

with "http://" indicating the protocol, "server.host.domain" uniquely identifying the server, and "directory/subdir/file" identifying the document to be served. Where the document is not to be accessible to other than an authorized client, the URL might be of a form similar to the sessionless URL:

http://server.host.domain/dir...file?userfred+password

In the session-specific URL, the user's name and password are included in the URL. While this is useful for making a reference to a single document for a specific user, it is not useful for passing sensitive session information because, the user name and password being in plaintext, it is too easily tampered with.

Thus, what is needed is a method and apparatus for maintaining a seamless and secure migration of a session between a client and a first server to a session between the client and a second server, where the first and second servers are independently controlled.

SUMMARY OF THE INVENTION

An improved session control method and apparatus is provided by virtue of the present invention. In one embodiment of the present invention, a client establishes a session with a first server such that the first server can identify the client. When the client wishes to migrate from the first server to a second server, the client requests a session token from the first server. The session token is a data element generated by the first server which is unique over the client-server network being navigated and identifies the particular session with the first server. The session token is preferably a difficult to forge data element, such as a data element digitally signed using the private key of the first server and possibly encrypted. The session token might also be encrypted, using public key encryption or other methods, to prevent the session token from being easily readable.

The session token is passed from the client to the second server to initiate migration to the second server. In some embodiments, session data is too bulky to be passed as part of the session token. In such cases, the second server uses the session token to formulate a request to the first server for the data needed to handle the state of the session and the first server provides the data to the second server. To minimize the transmission of data, the second server might maintain a version of the bulk session data and only request an update to the version of the data indicated in the session token.

In the preferred embodiment, the token is passed in a URL (Uniform Resource Locator). However, it also might be passed as an HTTP "cookie".

A further understanding of the nature and advantages of the inventions herein may be realized by reference to the remaining portions of the specification and the attached drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram of a client-server network with which the present invention is used.

FIG. 2 is a flowchart of a session migration process according to the present invention.

FIG. 3 is a schematic of a session token.

FIG. 4 is a schematic of a request for out-of-band data.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

FIG. 1 illustrates an overall structure of a client-server network 10 with which the present invention is used. Network 10 is shown comprising a client 12, two servers 14A, 14B, the Internet 16, server connections 18A, 18B to Internet 16 and a client connection 18C to Internet 16. As should be apparent, more than two servers might be coupled by a network to one or more clients, and the network need not be the global Internet, but can be any other network of cooperating independent servers. One alternative platform for the present invention is an intranet rather than Internet, where the intranet is an internal network within an organization using Internet protocols to communicate internal data. Depending on the size of the organization, the intranet might comprise several cooperating, but independent servers.

In operation, server 14A interacts with client 12 via Internet 16 by sending and receiving data packets directed at, and received from, client 12. Client 12 also sends packets to and receives packets from server 14A via Internet 16. Thus, at a network level higher than the transport level, there exists a logical channel 20A connecting client 12 to server 14A. The equivalent can be said about server 14B, which connects to client 12 via a logical channel 20B.

Server 14A and 14B are cooperative servers in that they both understand that client 12 might migrate from one server to another and facilitate that process. Unlike a centralized access control system, however, servers 14A and 14B are separately controlled.

An example of session migration will now be described with reference to a migration of client from a server A to a server B, which might be client 12, server 14A (source server node) and server 14B (target server node). The session migration process begins when client 12 takes an action which is to result in a session migration. The session migration moves the client's interaction from channel 20A to channel 20B in a way that is transparent to the client's user, but that also maintains any necessary state, including state variables which indicate a level of authorization for the client's use of the session.

One instance where migration is useful is where an operator of a Web server wants to out source the operation of a particular aspect of their Web site. For example, a newspaper might operate a site which provides news, features and classifieds, but will out source the management of the classifieds to another operator, such as Electric Classifieds, Inc., of San Francisco, Calif. (the assignee of the present application). A migration occurs when a user selects a classifieds "button" or selection on a Web page operated by the newspaper. The URL associated with the button can either be a direct reference to the target server or a reference which causes redirection. In the former case, the page containing the button is presented to the user with the underlying HTML anchor including a session token for use in migration. This is possible where the source server can anticipate that a migration may occur. In the latter case, the server need not anticipate the migration and the anchor for the button will be a URL directed at the source server. The source server responds to the URL with a redirection URL. The redirection URL would be the equivalent to the embedded URL with a session token in the former case.

A session migration using a redirection URL is illustrated in the flowchart of FIG. 2. FIG. 2 shows a number of steps identified as S1, S2, S3, etc. which are performed in ascending numerical order unless otherwise indicated. Before the migration process begins, the client is in a session with server A (step S1). The client sends requests to server A and server A services those requests. How server A services those requests depends on what the request is and the current state of the session. For example, for some requests, server A will only respond to the request if the state of the session with the client indicates that the client is logged in as an authorized user. The state might also include variables which modify the interaction with the client, such as the client's user's demographics. If the demographics indicate the geographic location of the user or the users age, gender, etc., the server might modify advertisements presented to the client accordingly. If the demographics indicate that the user is a sophisticated user, the server might send less help text in exchange for quicker transmission times. When such state variables are used, it should be apparent from this description that the server to which the client is migrating needs to have the same state variables so as to present a consistent style across the migration. State variables might also indicate what actions the client has already done, to allow for activity tracking and caching. Activity tracking is useful for Web browser clients, so that the navigation of the client can be tracked. Caching is useful for many types of clients and might be used to avoid sending data the server knows the client already has.

In step S2, the client makes a selection which server A determines will be migrated. In the case of embedded migration (not shown), server A would have anticipated the possible migration and included the migration information in an HTML anchor on the page. Server A optionally verifies if the client is permitted to migrate and then server A returns (step S3) a session token 104 to the client as part of a URL. By including the session token as part of the URL, a standard browser (i.e., one which is not aware of migration capabilities) can be used. The structure of a session token is shown in Table 1 and illustrated in FIG. 3.

TABLE 1. SESSION TOKEN ELEMENTS

1. Server Node Identifier
2. Unique Session Identifier for the Server Node
3. Time of session creation
4. Expiration (Time-to-live) of session
5. User ID
6. E-mail address
7. Telephone number
8. Postal address
9. Digital Signature

The server node identifier can be almost any type of identifier, including numerical values or ASCII strings. The server node identifier might be an identifier specific to cooperating servers or could be a unique code already in use by the server such as its four-byte IP address or six-byte IP6 address. The only requirement is that no two cooperating server nodes share the same identifier. The session is uniquely identified by the first two elements in the session token, since the second element is unique within the source server node. By dividing the session identifier into two pieces, we allow for the decentralized generation of session identifiers. Some of the session token elements, such as the e-mail address, telephone number and postal address, might be optional elements.

As shown in FIG. 3, a session token such as session token 104 is divided into a plaintext portion 300 containing the session ID and a ciphertext portion containing the encrypted session information 302 to allow for more secure transport

of session token information. With public key encryption, the information can be made secure by digitally signing it, to prevent forgery, or by encrypting it, to prevent forgery and to prevent others from reading the session data. In Table 1, the encrypted session information is the third through eighth elements, which are not required to uniquely identify the session. Therefore, these fields can be sent encrypted yet be decrypted at the target server node using an encryption method and key specific to the session. If the encryption method and key is the same for all session tokens from a particular source server node, the second element can also be encrypted. Encryption of the session specific information is preferred, so that unauthorized parties cannot view the information. In some embodiments, the information in the session token is not readable by the client even when the client is in possession of the session token. Of course, the user might be able to view his or her own session information through the use of a dummy server. A dummy server accepts the user's migration like any other server, but then does nothing more than display the session information.

Table 1 shows but one example of a format for a session token. After reading this description, the person of ordinary skill will understand that other formats are possible, so long as the format have certain properties. For example, session tokens must be unique so as to uniquely identify a session and must be such that they can be generated in a distributed environment, i.e., without a centralized session token generator. They should be difficult for unauthorized parties to modify or forge. They must also be able to pass information about the session, or at least refer to information about the session, which may then later be transferred out-of-band. While the system may operate without digital signatures, the use of digitally signed session tokens is preferred. A digitally signed session token is signed using the source server node's private key so that the digital signature can be verified using the source server node's public key. Using a digital signature allows the authenticity of the token to verified.

Once the client receives the session token from server A, the client sends session token **104** to the target server node, server B (step S4). Server B processes session token **104** (S5), performing the necessary verification and decrypting of the session token. In some cases, the session token makes reference to "out-of-band" data, i.e., data which is not contained within the session token but which is state information needed for migration. Out-of-band data is used to keep session tokens from being too large, or for tighter access control. For example, the user's postal address might be specified as a pointer to a specific record of a database controlled by server A. Server B checks to determine if out-of-band data is needed (S6). If out-of-band data is needed, server B forms a request **106** for the data using session token **104** and sends request **106** to server A (S7). Session token **104** is part of the request so that server A can verify that server B is making an authorized request. FIG. **4** shows the structure of request **106**, which is an envelope containing request details **400** and the session token **104**. In some embodiments, not all the session information is passed to the source server node.

When server A receives request **106**, it supplies the out-of-band data to server B (S8) in a response **108**. Server B parses and stores response **108** (S9) and proceeds to step S10. If out-of-band data is not needed, server B proceeds directly from step S6 to step S10. At step S10, server B decides whether to accept the server migration request represented by session token **104**. If server B decides not to honor the migration because the identified client is not authorized to use server B, server B does not support that particular migration, or any other reasons, server B sends an error message **110** back to the client. Otherwise, server B processes the information in session token **104** and out-of-band data **108**, if used, and sends an acceptance message **112** to the client (S11). Acceptance message **112** can either be a server to client message or a server to user message such as a welcome message. In the former case, the transition might be a transparent transition. The client responds (S12) by transitioning to the new session and requesting services from server B (S13) which server B services using the migrated state information (S14).

Versioning

To minimize the retransmission of data that a target server node already has, the target server node may choose to cache out-of-band data obtained from a source server node. Of course, in a distributed network, cache consistency can be a problem. Cache inconsistencies might occur when session information changes and a client is not connected to the target server node caching the data. It is important for those changes to work their way around to other server nodes.

A method for doing this is to encode a version identifier within the session token such that whenever any out-of-band type session information changes, the source server node will change the version identifier. Then, whenever a new session token is generated by the source server node, the new version identifier will be passed on to a target server node. If the version identifier within the token indicates that other information newer than the data that the target server node previously has used, the target server should request new out-of-band session information from the source server node.

Different granularities of version numbers may be used for out-of-band session information. For example, a single version number may represent the state of all out-of-band data items, or multiple version numbers may represent certain subsets of the out-of-band data. The granularity does not affect the cache updating per se. Instead, the granularity of versioning merely determines how much data must be re-transferred in the case of the change of session information.

One application for the present invention is within the World Wide Web, where session tokens are encoded within URLs. When a session is to be migrated from a source server node to a target server node, a new URL will be generated for a session token. A client application will then make a request to the target server node using this URL. The target server node will then decode this session token, verify its authenticity (using public key cryptography) and obtain any necessary out-of-band session information before continuing with the request. Preferably, the session token is limited in size so that most browsers and HTTP handlers can correctly process the session token as a URL. A limit many browsers have for URLs is 1 kilobyte. If the session information is greater than the limit, some of the session information is sent as out-of-band data.

In the preferred embodiment, the encoded session token will be in a modified Base-64, which is a standard encoding for MIME email documents (Multipurpose Internet Mail Extensions). A Base-64 data stream is a stream of 6-bit numbers represented by an alphabet of 64 alphanumeric characters (usually the uppercase letters A–Z, the lowercase letters a–z, the digits 0–9, and the characters '+' and '/'. For our purposes, we use a modified Base-64 wherein '–' and '_' are used in place of '+' and '/' so as to generate HTTP compliant URL's.

Before it is encoded into modified Base-64 for transmission using HTTP, the session token has the format shown in Table 2.

TABLE 2

Web Session Token Format

Bytes	Description
1	Token Format ID
8	Session Identifier
2	Length (N) of data to follow
N	Session Data
4	CRC Checksum (32 bit)

The Token Format ID should be zero. This field is reserved as a way to specify new token formats should there be a need to change them in the future. The session identifier is a 64-bit ID comprising 16 bits (2 bytes) for a server node ID and 48 bits (6 bytes) for a unique session ID for the server node. This assumes a cooperating network of 2^16 servers or fewer each assigned a unique server node ID. The N bytes of session data is encrypted with target server node's public key and digitally signed with source server node's private key. Encryption is done using a public key cryptosystem supplied by RSA Data Security, of Palo Alto, Calif., or similar cryptosystem.

Before the session data is encrypted and signed, it has the following format, which is very similar to the RFC 822 electronic mail header format:

<KEY1>: <VALUE1>

<KEY2>: <VALUE2>

...

<KEYM>: <VALUEM>

In this format, a key and its associated value are separated by a colon (and optional whitespace) and key/value pairs are separated by newline characters. An example of session data is as follows:

Version-Date: Fri Dec 30 10:14:46 PST 1994

User-ID: Fred

Email-Address: fred@fred.org

Telephone: @123.Fred.Telephone

Postal-Address: @123.Fred.Address

The "Telephone:" key value, "@123.Fred.Telephone", is an indirect reference to session information which indicates that Fred's telephone number is actually held at server node **123** and should be queried from that server node, using the data identifier "Fred.Telephone". To implement the out-of-band transferal of session information, a simple transfer protocol for making queries over a secure out-of-band channel is used. The secure out-of-band channel (depending upon the circumstances) may be implemented as a private, dedicated physical link (such as a dedicated, private T1 line), or may be implemented as a secure link over a public network using an encrypted tunnel (such as with SSL, Secure Sockets Layer).

A querying server node wishing to obtain session information from a source server node would simply open a connection with the source server node, identify itself (the querying server node), get authenticated and then make a set of queries to obtain session data. The following is an example of the process where a querying server node (identified as server node **456** in this example) connects to a source server node (identified as server node **123**) to request out-of-band data:

Server	Output
123	Connected to server 123, please identify and authenticate self
456	Server: 456 Signature REOp5XtI5XtIpWx1tBY6maOwRXuyNz
123	Server 456 authenticated, please make your requests.
456	Fred.Telephone, Fred.Address
123	Fred.Telephone: (345) 555-1212 Fred.Address: 12345 Main Street Anytown, USA 98765-1234
456	QUIT
123	Goodbye, server 456.

In addition to the above-described applications, other applications can be found for the present invention. One use of transparent session migration is a child filter. If a child is logged into a service, that child may see different information than if an adult were logged into the same service. As the child migrates from server to server, an indication that the user is a child will be included in the session information so that the target server node can adjust its content accordingly.

The above detailed description explains how to make and use a session migration method and apparatus according to the present invention. The above description is illustrative and not restrictive. Many variations of the invention will become apparent to those of skill in the art upon review of this disclosure. The scope of the invention should, therefore, be determined not with reference to the above description, but instead should be determined with reference to the appended claims along with their full scope of equivalents.

What is claimed is:

1. In a client-server network having multiple independent servers, a method of session migration from a session between a client and a first server to a session between the client and a second server;

establishing a current session between the client and the first server after the first server has verified that th e client is an authorized client of the first server;

sending a migration request from the client to the first server;

in response to the migration request generating a session token;

sending a session token from the first server to the client, wherein the session token uniquely identifies the current session between the client and the first server;

sending the session token from the client to the second server as a request to the second server for migration;

verifying, at the second server and from the session token, the identity of the client and the first server;

requesting, from the first server, data about the session, wherein the request is sent from the second server;

responding from the first server to the request for data by sending the requested data to the second server; and

if the client is an authorized client with respect to the second server, continuing the session with the second server in place of the first server.

2. The method of claim **1**, wherein the client-server network is the Internet and the first and second servers are hypertext transfer protocol servers.

3. The method of claim **1**, wherein the step of generating a session token is a step of generating an encrypted session token.

4. The method of claim 3, wherein the step of generating an encrypted session token comprises the steps of encrypting portions of the session token and including a digitally signed portion in the session token, the digitally signed portion being digitally signed by a private key of the first server.

5. The method of claim 1, wherein the step of sending a session token from the first server to the client is a step of sending a session token containing a digitally signed portion, the digitally signed portion being digitally signed by a private key of the first server.

6. The method of claim 1, further comprising the steps of:

requesting a password from the client;

sending the password to the first server;

if the password matches a predetermined criterion indicating an authorized client, continuing with the step of establishing a session;

sending, as part of the session token, an indication from the first server, that the client is an authorized client; and

using the indication from the first server as a substitute for a logon procedure used by the second server to verify the authorization of the client.

7. The method of claim 1, further comprising a step of forming a uniform resource locator representing the session token.

8. The method of claim 1, wherein the step of generating the session token is performed without a central token repository yet is guaranteed to be a unique session token.

9. The method of claim 1, further comprising a step of transferring the requested data using a protocol other than a hypertext transfer protocol.

10. The method of claim 1, further comprising a step of maintaining a version control number and requesting from the first server data which is an update to data received previously at the second server, the update being an update from a previous version of the data identified by a version number stored on the second server and a current version of the data identified by a version number stored on the first server.

11. The method of claim 1, wherein the session migration is from one World Wide Web server to a second World Wide Web server, and the client is a World Wide Web browser.

12. The method of claim 1, wherein each step of sending a session token comprises sending at least a part of the session token in a uniform resource locator.

13. The method of claim 1, wherein each step of sending a session token comprises sending at least a part of the session token in a hypertext transport protocol cookie.

* * * * *

L Number	Hits	Search Text	DB	Time stamp
-	63	(session near8 identif$4) and (digest near11 (messag$4 key$4))	USPAT; EPO; JPO; DERWENT; IBM_TDB	2003/03/23 18:32
-	22	((server near11 (multipl$4 pluralit$4 many)) and (session near8 identif$4) and (second near11 server)) and (digest near11 (messag$4 key$4))	USPAT; EPO; JPO; DERWENT; IBM_TDB	2003/03/20 15:59
-	428	(server near11 (multipl$4 pluralit$4 many)) and (session near8 identif$4) and ((second different another) near11 server)	USPAT; EPO; JPO; DERWENT; IBM_TDB	2003/03/20 17:46
-	15	((server near11 (multipl$4 pluralit$4 many)) and (session near8 identif$4) and ((second different another) near11 server)) and (digest$4 near11 (messag$4 key$4)) and (secur$4 near8 session) and (stat$4 near8 (information data packet$4 bit$4)) and (ssl (secur$4 adj socket$4 adj layer)) and databas$4	USPAT; EPO; JPO; DERWENT; IBM_TDB	2003/03/20 16:35
-	19	(server near11 (multipl$4 pluralit$4 many)) and (session near8 identif$4) and (digest$4 near11 (messag$4 key$4)) and (secur$4 near8 session) and (stat$4 near8 (information data packet$4 bit$4)) and (ssl (secur$4 adj socket$4 adj layer)) and (stat$4 near11 databas$4) and ((send$4 forward$4 transmi$8 transfer$6 receiv$4 retriev$4) near11 stat$4 near11 (information data packet$4 fram$4 bit$4 fil$4))	USPAT; EPO; JPO; DERWENT; IBM_TDB	2003/03/20 17:48
-	6	((server near11 (multipl$4 pluralit$4 many)) and (session near8 identif$4) and ((second different another) near11 (server connection))) and (stat$4 near11 (information data packet$4 bit$4) near11 databas$4) and (ssl (secur$4 adj socket$4 adj layer)) and ((send$4 forward$4 transmi$8 transfer$6 receiv$4 retriev$4) near11 stat$4 near11 (information data packet$4 fram$4 bit$4 fil$4)) not (((server near11 (multipl$4 pluralit$4 many)) and (session near8 identif$4) and ((second different another) near11 (server connection))) and (digest$4 near11 (messag$4 key$4)) and (stat$4 near11 (information data packet$4 bit$4) near11 databas$4) and (ssl (secur$4 adj socket$4 adj layer)) and ((send$4 forward$4 transmi$8 transfer$6 receiv$4 retriev$4) near11 stat$4 near11 (information data packet$4 fram$4 bit$4 fil$4)))	USPAT; EPO; JPO; DERWENT; IBM_TDB	2003/03/20 18:11
-	84	((server near11 (multipl$4 pluralit$4 many)) and (session near8 identif$4) and ((second different another) near11 server)) and (secur$4 near8 session)	USPAT; EPO; JPO; DERWENT; IBM_TDB	2003/03/21 08:36
-	482	(server near11 (multipl$4 pluralit$4 many)) and (session near8 identif$4) and ((second different another) near11 (server connection))	USPAT; EPO; JPO; DERWENT; IBM_TDB	2003/03/21 08:36

-	5	(server near11 (multipl$4 pluralit$4 many)) and (session near8 identif$4 near11 stat$4) and ((second different another) near11 (server connection)) and (ssl (secur$4 adj socket$4 adj layer)) and (secur$4 near8 session) not (((server near11 (multipl$4 pluralit$4 many)) and (session near8 identif$4) and ((second different another) near11 (server connection))) and (digest$4 near11 (messag$4 key$4)) and (stat$4 near11 (information data packet$4 bit$4) near11 databas$4) and (ssl (secur$4 adj socket$4 adj layer)) and ((send$4 forward$4 transmi$8 transfer$6 receiv$4 retriev$4) near11 stat$4 near11 (information data packet$4 fram$4 bit$4 fil$4)))	USPAT; EPO; JPO; DERWENT; IBM_TDB	2003/03/23 18:21
-	9	(server near11 (multipl$4 pluralit$4 many)) and (session near8 identif$4 near11 stat$4) and ((second different another) near11 (server connection)) and (ssl (secur$4 adj socket$4 adj layer)) not (((server near11 (multipl$4 pluralit$4 many)) and (session near8 identif$4) and ((second different another) near11 (server connection))) and (digest$4 near11 (messag$4 key$4)) and (stat$4 near11 (information data packet$4 bit$4) near11 databas$4) and (ssl (secur$4 adj socket$4 adj layer)) and ((send$4 forward$4 transmi$8 transfer$6 receiv$4 retriev$4) near11 stat$4 near11 (information data packet$4 fram$4 bit$4 fil$4)))	USPAT; EPO; JPO; DERWENT; IBM_TDB	2003/07/09 15:50
-	1	(server near11 (multipl$4 pluralit$4 many) same stat$4 near8 (information data packet$4 bit$4) near11 identif$5) and (ssl (secur$4 adj socket$4 adj layer)) not bowman$7.in.	USPAT; EPO; JPO; DERWENT; IBM_TDB	2003/03/21 13:35
-	19	((server near11 (multipl$4 pluralit$4 many)) and (session near8 identif$4) and ((second different another) near11 (server connection))) and (digest$4 near11 (messag$4 key$4)) and (stat$4 near11 (information data packet$4 bit$4) near11 databas$4) and (ssl (secur$4 adj socket$4 adj layer)) and ((send$4 forward$4 transmi$8 transfer$6 receiv$4 retriev$4) near11 stat$4 near11 (information data packet$4 fram$4 bit$4 fil$4))	USPAT; EPO; JPO; DERWENT; IBM_TDB	2003/03/21 17:01
-	57	(session near8 identif$4) and (messag$4 adj2 digest)	USPAT; EPO; JPO; DERWENT; IBM_TDB	2003/07/09 15:33
-	9	((server near11 (multipl$4 pluralit$4 many)) and (session near8 identif$4) and ((second different another) near11 server)) and (digest$4 near11 (messag$4 key$4)) and (secur$4 near8 session) and (stat$4 near8 (information data packet$4 bit$4)) and (ssl (secur$4 adj socket$4 adj layer)) and (stat$4 near11 databas$4)	USPAT; EPO; JPO; DERWENT; IBM_TDB	2003/07/09 15:38
-	24	(server near11 (multipl$4 pluralit$4 many) same stat$4 near8 (information data packet$4 bit$4)) and (ssl (secur$4 adj socket$4 adj layer)) not bowman$7.in.	USPAT; EPO; JPO; DERWENT; IBM_TDB	2003/07/09 15:48
-	22	((server near11 (multipl$4 pluralit$4 many)) and (session near8 identif$4) and (second near11 server)) and (digest$4 near11 (messag$4 key$4)) and (secur$4 near8 session)	USPAT; EPO; JPO; DERWENT; IBM_TDB	2003/11/05 12:46

-	20	((server near11 (multipl$4 pluralit$4 many)) and (session near8 identif$4) and ((second different another) near11 server)) and (digest$4 near11 (messag$4 key$4)) and (secur$4 near8 session) and (stat$4 near8 (information data packet$4 bit$4)) and (ssl (secur$4 adj socket$4 adj layer))	USPAT; EPO; JPO; DERWENT; IBM_TDB	2003/11/05 14:59
-	120	(server near11 (multipl$4 pluralit$4 many)) and (session near8 identif$4) and ((second different another) near11 (server connection)) and (secur$4 near8 session)	USPAT; EPO; JPO; DERWENT; IBM_TDB	2003/11/05 15:55
-	61	((server near11 (multipl$4 pluralit$4 many)) and (session near8 identif$4) and ((second different another) near11 (server connection)) and (secur$4 near8 session)) and (((maintain$4 "same") with (key public privat$4 cryptograph$4 encrypt$6)) and (new$3 with (connection shar$4 session)))	USPAT; EPO; JPO; DERWENT; IBM_TDB	2003/11/05 16:03
-	55	(((server near11 (multipl$4 pluralit$4 many)) and (session near8 identif$4) and ((second different another) near11 (server connection)) and (secur$4 near8 session)) and (((maintain$4 "same") with (key public privat$4 cryptograph$4 encrypt$6)) and (new$3 with (connection shar$4 session)))) not accenture.as.	USPAT; EPO; JPO; DERWENT; IBM_TDB	2003/11/05 16:03
-	40	((server near11 (multipl$4 pluralit$4 many)) and (session near8 identif$4) and ((second different another) near11 (server connection)) and (secur$4 near8 session)) and (((maintain$4 "same") with (key public privat$4 cryptograph$4 encrypt$6) same (connection shar$4 session)))	USPAT; EPO; JPO; DERWENT; IBM_TDB	2003/11/05 16:03
-	63	((ssl (secur$4 adj socket$4 adj layer)) same session same (maintain$4 "same") same (server firewall gateway prox$4))	USPAT; EPO; JPO; DERWENT; IBM_TDB	2004/04/18 20:41
-	28	((server near11 (multipl$4 pluralit$4 many)) and (session near8 identif$4) and (second near11 server)) and (digest$4 near11 (messag$4 key$4))	USPAT; EPO; JPO; DERWENT; IBM_TDB	2003/11/05 17:33
-	47	(server near11 (multipl$4 pluralit$4 many)) and ((second different another) near11 server) and (ssl (secur$4 adj socket$4 adj layer)) and (stat$4 near8 (information data packet$4 bit$4) near11 identif$5)	USPAT; EPO; JPO; DERWENT; IBM_TDB	2004/04/18 20:38
-	21	((server near11 (multipl$4 pluralit$4 many)) and (session near8 identif$4) and ((second different another) near11 server)) and (digest$4 near11 (messag$4 key$4)) and (secur$4 near8 session) and (stat$4 near8 (information data packet$4 bit$4))	USPAT; EPO; JPO; DERWENT; IBM_TDB	2003/11/05 17:33
-	26	((server near11 (multipl$4 pluralit$4 many)) and (session near8 identif$4) and ((second different another) near11 server)) and (digest$4 near11 (messag$4 key$4)) and (secur$4 near8 session)	USPAT; EPO; JPO; DERWENT; IBM_TDB	2003/11/05 17:33
-	35	(server near11 (multipl$4 pluralit$4 many)) and ((second different another) near11 server) and (ssl (secur$4 adj socket$4 adj layer)) and (stat$4 near8 (information data packet$4 bit$4) near11 identif$5) and ((session connect$6) with establ$6 with (reus$4 exist$4 "same"))	USPAT; EPO; JPO; DERWENT; IBM_TDB	2004/04/18 20:39

-	7	((server near11 (multipl$4 pluralit$4 many)) and ((second different another) near11 server) and (ssl (secur$4 adj socket$4 adj layer)) and (stat$4 near8 (information data packet$4 bit$4) near11 identif$5) and ((session connect$6) with establ$6 with (reus$4 exist$4 "same"))) not bowman$6.in.	USPAT; EPO; JPO; DERWENT; IBM_TDB	2004/04/18 20:39
-	17	((ssl (secur$4 adj socket$4 adj layer)) same session same (maintain$4 "same") same (server firewall gateway prox$4)) and ((session connect$6) with establ$6 with (reus$4 exist$4 "same"))	USPAT; EPO; JPO; DERWENT; IBM_TDB	2004/04/18 21:04
-	291	(ssl (secur$4 adj socket$4 adj layer)) and ((second another) with (server firewall gateway prox$4)) and ((session connect$6) with (reus$4 exist$4 maintain$4 "same"))	USPAT; EPO; JPO; DERWENT; IBM_TDB	2004/04/18 21:08
-	270	(ssl (secur$4 adj socket$4 adj layer)) and ((second another) with (server prox$4)) and ((session connect$6) with (reus$4 exist$4 maintain$4 "same"))	USPAT; EPO; JPO; DERWENT; IBM_TDB	2004/04/18 21:10
-	37	(ssl (secur$4 adj socket$4 adj layer)) and ((second another) with (server prox$4)) same ((session connect$6) with (reus$4 exist$4 maintain$4 "same"))	USPAT; EPO; JPO; DERWENT; IBM_TDB	2004/04/18 21:10
-	191	(ssl (secur$4 adj socket$4 adj layer)) and ((second another) near3 (server prox$4)) and ((session connect$6) with (reus$4 exist$4 maintain$4 "same"))	USPAT; EPO; JPO; DERWENT; IBM_TDB	2004/04/18 21:19
-	16	(ssl (secur$4 adj socket$4 adj layer)) and ((second another) near3 (server prox$4)) same ((session connect$6) with (reus$4 exist$4 maintain$4 "same"))	USPAT; EPO; JPO; DERWENT; IBM_TDB	2004/04/18 21:14
-	16	(ssl (secur$4 adj socket$4 adj layer)) and ((second$4 another) near3 (server prox$4)) same ((session connect$6) with (reus$4 exist$4 maintain$4 "same"))	USPAT; EPO; JPO; DERWENT; IBM_TDB	2004/04/18 21:15
-	105	(ssl (secur$4 adj socket$4 adj layer)) and (secure near4 connect$5) and ((second another) near3 (server prox$4)) and ((session connect$6) with (reus$4 exist$4 maintain$4 "same"))	USPAT; EPO; JPO; DERWENT; IBM_TDB	2004/04/18 21:19